NASA/TP-1998-206949

# Algorithms for Efficient Computation of Transfer Functions for Large Order Flexible Systems

*Peiman G. Maghami and Daniel P. Giesy*
*Langley Research Center, Hampton, Virginia*

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

March 1998

# Contents

## List of Figures

## List of Tables

## Abstract

*An efficient and robust computational scheme is given for the calculation of the frequency response functions of a large order, flexible system implemented with a linear, time invariant control system. Advantage is taken of the highly structured sparsity of the system matrices based on a model of the structure using normal mode coordinates. The computational time per frequency point of the new computational scheme is a linear function of system size, a significant improvement over traditional, full-matrix techniques whose computational times per frequency point range from quadratic to cubic functions of system size. This permits the practical frequency domain analysis of systems of much larger order than by traditional, full-matrix techniques. Formulations are given for both open- and closed-loop systems. Numerical examples are presented showing the advantages of the present formulation over traditional approaches, both in speed and in accuracy. Using a model with 703 structural modes, the present method was up to two orders of magnitude faster than a traditional method. The present method generally showed good to excellent accuracy throughout the range of test frequencies, while traditional methods gave adequate accuracy for lower frequencies, but generally deteriorated dramatically in performance at higher frequencies.*

# 1. Introduction

Control of flexible systems has received significant attention in the literature. To date, numerous techniques, algorithms and procedures have been developed for design of controllers for such systems ranging from spacecraft and satellites to aircraft, ships, machines, etc. These flexible systems which are generally infinite-dimensional are typically modeled using a finite number of generalized coordinates or modes. Control of flexible systems may become difficult depending on the number, location, relative proximity, and inherent damping of these modes. The response of the system to a given disturbance/excitation generally depends on modal properties (amplitude, frequency, and damping) and the amplitude and phase content of the disturbance/excitation. In general, two techniques, time domain analysis and frequency domain analysis, have been developed and extensively used to analyze and characterize the input/output behavior of linear time-invariant systems including flexible systems. In frequency domain analysis, frequency response functions (defined as transfer function matrices from inputs to outputs of the system) have typically been used (usually in the form of magnitude and phase or Bode plots) in the analysis of linear systems as well as in designing controllers for such systems. In general, frequency response functions of the open-loop system are used to evaluate the performance of the open-loop system, and to identify and quantify needed performance and/or stability improvements at various frequency bands. The closed-loop frequency response functions are typically needed to insure that desired performance and stability have been achieved by the control system. Moreover, frequency-domain specifications such as peak magnitude, bandwidth, roll-off rate, etc. are often used in characterizing the desired behavior of the system in the frequency domain (this is known as loop shaping).

In general, the order of the flexible system (as defined by the number of modes retained in the model) for which open-loop and/or closed-loop analysis is performed depends on the application considered. For example, if the closed-loop response of a spacecraft with a low-bandwidth attitude control system is of interest, then a small set of modes would be sufficient to capture the low frequency closed-loop behavior of the system. On the other hand, if the response of the flexible system is desired over a large frequency range or if the control system considered has a high bandwidth, then a large set of modes (in the hundreds or thousands) may be necessary to capture the true response of the system.

However, the current techniques for obtaining frequency response functions, although able to deal with small or medium size systems, have problems in handling large order systems. A straightforward calculation of the frequency response function matrix at a single frequency point which is based on the definition of the transfer function has a computational cost which is a cubic function of the system size. If this calculation must be repeated for many frequency points, Laub ([1, 2]) presents a technique which has a better average cost. This technique performs an initial orthogonal transformation of the system which reduces the system response matrix to upper Hessenberg form. This initial transformation has a computational cost which is a cubic function of the system size. This technique can then calculate the frequency response function matrix at each frequency point at a cost which is a quadratic function of the system size. However, for very large systems (many hundreds of modes or more), even this is too slow, and a better method is needed.

To this end, this paper describes efficient techniques for the computation of open- and closed-loop frequency response functions of large order flexible systems. The proposed techniques are computationally robust and accurate. The closed-loop frequency response function calculation is novel and constitutes enabling technology for the frequency domain analysis of large flexible systems. These techniques take advantage of the sparsity of the flexible systems in normal mode coordinates and reduce the computational cost from a quadratic function of the order of the system to a linear function. A fringe benefit of these faster computational techniques is an improvement in accuracy. The decoupling of the calculations involving each structural mode not only makes the calculation faster, but also means that lower order, better conditioned linear systems are being solved; and this improves accuracy. Numerical examples are presented showing the advantages of the present formulation over traditional approaches, both in speed and in accuracy. The present paper expands on the authors' previous work [3].

## Symbols

| | |
|---|---|
| $A$ | generic matrix |
| $A_c$ | $k \times k$ control system state matrix |
| $A_s$ | $2p \times 2p$ plant state matrix for first-order model, see Equation (3) |
| $A_s^i$ | $2 \times 2$ diagonal block $i$ of $A_s$ $1 \leq i \leq p$; see Equation (4) |

| | |
|---|---|
| $\widetilde{A}$ | $(2p+r) \times (2p+r)$ system matrix of the system which combines both the plant and the controller |
| $B_c$ | $k \times f$ matrix of tracking error (resp., reference input) influence on closed-loop (resp., open-loop) control system states |
| $B_s$ | $2p \times m$ matrix of control input influence on first-order model states, see Equation (5) |
| $B_w$ | $2p \times g$ matrix of disturbance input influence on first-order model states |
| $\widetilde{B}$ | any of the block matrix columns in Equation (31) |
| $b$ | generic scalar or vector |
| $b_s^{(i)}$ | row $i$ of matrix $B_s$ |
| $C_c$ | $m \times k$ matrix of control system state influence on negative of control input vector |
| $C_p$ | $f \times n$ matrix of influence of states of second-order physics based model on measurement output |
| $\bar{C}_p$ | $f \times p$ matrix of influence of modal model states on measurement output |
| $\bar{C}_p(i,j)$ | $(i,j)$ element of matrix $\bar{C}_p$ |
| $C_r$ | $f \times n$ matrix of influence of state rates of second-order physics based model on measurement output |
| $\bar{C}_r$ | $f \times p$ matrix of influence of modal model state rates on measurement output |
| $\bar{C}_r(i,j)$ | $(i,j)$ element of matrix $\bar{C}_r$ |
| $C_s$ | $f \times 2p$ matrix of influence of first-order model states on measurement output, see Equation (6) |
| $C^{pr}$ | $l \times 2p$ matrix of influence of states of first-order model with input feedthrough on performance outputs |
| $C_a^{pr}$ | $l \times n$ matrix of influence of state accelerations of second-order model on performance output |
| $\bar{C}_a^{pr}$ | $l \times p$ matrix of influence of modal model state accelerations on performance output |

| | |
|---|---|
| $C_p^{pr}$ | $l \times n$ matrix of influence of states of second-order model on performance output |
| $\bar{C}_p^{pr}$ | $l \times p$ matrix of influence of modal model states on performance output |
| $C_r^{pr}$ | $l \times n$ matrix of influence of state rates of second-order model on performance output |
| $\bar{C}_r^{pr}$ | $l \times p$ matrix of influence of modal model state rates on performance output |
| $C_1^{pr}$ | $l \times 2p$ matrix of influence of states of first-order model with acceleration term on performance outputs |
| $C_2^{pr}$ | $l \times 2p$ matrix of influence of state rates of first-order model with acceleration term on performance outputs |
| $\tilde{C}$ | generic input influence matrix of a linear system |
| $D$ | $n \times n$ damping matrix of second-order physics based model |
| $\bar{D}$ | $p \times p$ damping matrix of modal model |
| $D^{pr}$ | $l \times m$ feedthrough influence of control input on performance outputs in first-order model with input feedthrough |
| $D_w^{pr}$ | $l \times g$ feedthrough influence of disturbance input on performance outputs in first-order model with input feedthrough |
| $\tilde{D}$ | generic feedthrough matrix of a linear system |
| $d$ | $m \times 1$ input noise/distrubance vector |
| $E(s)$ | $(2p + k) \times (2p + k)$ matrix $sI - \tilde{A}$ |
| $E_{11}(s)$ | $2p \times 2p$ upper left submatrix of $E(s)$ |
| $E_{12}(s)$ | $2p \times k$ upper right submatrix of $E(s)$ |
| $E_{21}(s)$ | $k \times 2p$ lower left submatrix of $E(s)$ |
| $E_{22}(s)$ | $k \times k$ lower right submatrix of $E(s)$ |
| $e$ | $f \times 1$ tracking error vector |
| $f$ | number elements in measurement output vector |
| $G$ | generic matrix output of Matlab function `ltifr` |

| | |
|---|---|
| $G(s)$ | $(l+f)\times(m+g)$ open-loop transfer function matrix of first order plant model |
| $G_{11}(s)$ | $l \times m$ upper left submatrix of $G(s)$ |
| $G_{12}(s)$ | $l \times g$ upper right submatrix of $G(s)$ |
| $G_{21}(s)$ | $f \times m$ lower left submatrix of $G(s)$ |
| $G_{22}(s)$ | $f \times g$ lower right submatrix of $G(s)$ |
| $g$ | number of elements in exogenous disturbance vector |
| $H$ | $n \times m$ matrix of influence of control input in second-order model |
| $\bar{H}$ | $p \times m$ matrix of influence of control input in modal model |
| $\bar{H}_{ij}$ | the $(i,j)$ element of matrix $\bar{H}$ |
| $H_w$ | $n \times g$ matrix of influence of exogenous disturbance in second-order model |
| $\bar{H}_w$ | $p \times g$ matrix of influence of exogenous disturbance in modal model |
| $I$ | identity matrix of the proper size to make sense in context |
| $i$ | subscript to index the retained modes, $1 \leq i \leq p$ |
| $j$ | $\sqrt{-1}$, the imaginary unit for complex numbers |
| $K$ | $n \times n$ stiffness matrix of second-order model |
| $K(s)$ | $m \times f$ controller transfer function matrix |
| $\bar{K}$ | $p \times p$ stiffness matrix of modal model |
| $k$ | number of control system states |
| $l$ | number of elements in performance output vector |
| $M$ | $n \times n$ mass matrix of second-order model |
| $\bar{M}$ | $p \times p$ mass matrix of modal model |
| $m$ | number elements in control input vector |
| $n$ | number of states in second-order model |
| $p$ | number of normal modes retained for modal equations |
| $Q(s)$ | $2p \times m$ matrix $(sI - A_s)^{-1}B_s$ |
| $Q_i(s)$ | $2 \times p$ submatrix of $Q(s)$ containing rows number $2i-1$ and $2i$, $1 \leq i \leq p$ |

| | |
|---|---|
| $q$ | $p \times 1$ vector of modal coordinates |
| $q_i$ | component $i$ of vector $q$, $1 \leq i \leq p$ |
| $r$ | $f \times 1$ reference input vector |
| $s$ | complex variable used in transfer functions (Laplace transforms) |
| $T_s$ | sampling frequency of a discrete-time system |
| $T_{\alpha\beta}$ | transfer function from $\beta$ to $\alpha$, where $\alpha \in \{y, y^{pr}, e, u\}$ and $\beta \in \{r, d, w, v\}$ |
| $u$ | $m \times 1$ control input vector |
| $v$ | $f \times 1$ measurement noise vector |
| $W$ | generic matrix |
| $w$ | $g \times 1$ exogenous disturbance vector |
| $X$ | generic matrix |
| $X(s)$ | $(2p+k) \times (2p+k)$ matrix $E(s)^{-1}$ |
| $X_{11}(s)$ | $2p \times 2p$ upper left submatrix of $X(s)$ |
| $X_{12}(s)$ | $2p \times k$ upper right submatrix of $X(s)$ |
| $X_{21}(s)$ | $k \times 2p$ lower left submatrix of $X(s)$ |
| $X_{22}(s)$ | $k \times k$ lower right submatrix of $X(s)$ |
| $x$ | $n \times 1$ position/attitude vector of second-order model |
| $x$ | generic scalar or vector unknown in a linear equation |
| $x_a$ | generic approximate scalar or vector solution to a linear equation |
| $x_c$ | $k \times 1$ vector of control system states |
| $x_s$ | $2p \times 1$ state vector for first-order model, see Equation (2) |
| $Y$ | generic matrix |
| $y$ | $f \times 1$ vector of measurements outputs |
| $y^{pr}$ | $l \times 1$ vector of performance outputs |
| $z$ | generic vector |
| $\Delta(s)$ | $k \times k$ matrix $E_{22}(s) - E_{21}E_{11}^{-1}(s)E_{12}$ |
| $\delta$ | symmetric relative error function |
| $\zeta$ | generic complex number |

| | |
|---|---|
| $\zeta_i$ | open-loop damping ratio of retained mode $i$, $1 \leq i \leq p$ |
| $\eta$ | generic complex number |
| $\xi$ | scalar comparison magnitude for generic scalar $x$ |
| $\Phi$ | $n \times p$ matrix whose columns are the $p$ retained mode shapes$[\phi_1 \quad \phi_2 \quad \ldots \quad \phi_p]$ |
| $\phi_i$ | $n \times 1$ mode shape vector of retained mode $i$, $1 \leq i \leq p$ |
| $\omega$ | a scalar or vector of frequency values |
| $\omega_i$ | open-loop frequency of retained mode $i$, $1 \leq i \leq p$ |

# 2. Mathematical Formulation

## 2.1. Second-Order Modal Equations

The dynamics of a typical linear, time-invariant flexible system may be expressed in a second-order model as

$$\begin{gathered} M\ddot{x} + D\dot{x} + Kx = Hu + H_w w + Hd \\ y = C_p x + C_r \dot{x} \\ y^{pr} = C_p^{pr} x + C_r^{pr} \dot{x} + C_a^{pr} \ddot{x} \end{gathered}$$

where $M, D$, and $K$ are the $n \times n$ mass, damping and stiffness matrices, respectively; $x$ is the $n \times 1$ position/attitude vector; $u$ is the $m \times 1$ control input vector; $w$ is the $g \times 1$ exogenous disturbance vector; $d$ is the $m \times 1$ input noise/disturbance vector; $H$ is the $n \times m$ control input influence matrix; and $H_w$ is the $n \times g$ exogenous disturbance influence matrix. The vectors $y$ and $y^{pr}$ are, respectively, the $f \times 1$ measurements output vector and the $l \times 1$ vector of performance outputs; $C_p$ and $C_r$ are $f \times n$ measurement output influence matrices; and $C_p^{pr}$, $C_r^{pr}$, and $C_a^{pr}$ are $l \times n$ performance output influence matrices.

If the second-order system is transformed into normal mode coordinates, and $p$ of the normal modes are retained to capture the relevant dynamics of the structure, then the system equations may be written in a modal form as

$$\begin{gathered} \bar{M}\ddot{q} + \bar{D}\dot{q} + \bar{K}q = \bar{H}u + \bar{H}_w w + \bar{H}d \\ y = \bar{C}_p q + \bar{C}_r \dot{q} \\ y^{pr} = \bar{C}_p^{pr} q + \bar{C}_r^{pr} \dot{q} + \bar{C}_a^{pr} \ddot{q} \end{gathered}$$

where $\bar{M}$, $\bar{D}$, and $\bar{K}$ are the $p \times p$ modal mass, damping, and stiffness matrices, respectively; $q$ is the $p \times 1$ vector of modal coordinates; and $\bar{H}$ and $\bar{H}_w$ are the $p \times m$ control input and the $p \times g$ disturbance influence matrices in modal coordinates, respectively. The matrices $\bar{C}_p$ and $\bar{C}_r$ are $f \times p$ measurement output influence matrices in modal coordinates; and $\bar{C}_p^{pr}$, $\bar{C}_r^{pr}$, and $\bar{C}_a^{pr}$ are $l \times p$ performance output influence matrices in modal coordinates.

It is assumed that the mode shapes are normalized with respect to the mass matrix, and modal damping is assumed. This means that $\bar{M} = I$, $\bar{D} =$

$\operatorname{diag}\{2\zeta_1\omega_1, 2\zeta_2\omega_2, \ldots, 2\zeta_p\omega_p\}$, and $\bar{K} = \operatorname{diag}\{\omega_1^2, \omega_2^2, \ldots, \omega_p^2\}$ where $\omega_i$ and $\zeta_i$ are the open-loop frequencies and damping ratios.

The control input and disturbance influence matrices are given by:

$$\bar{H} = \Phi^T H$$
$$\bar{H}_w = \Phi^T H_w$$

The measurement and performance output influence matrices are given by:

$$\bar{C}_p = C_p\Phi \quad ; \quad \bar{C}_r = C_r\Phi$$
$$\bar{C}_p^{pr} = C_p^{pr}\Phi \quad ; \quad \bar{C}_r^{pr} = C_r^{pr}\Phi \quad ; \quad \bar{C}_a^{pr} = C_a^{pr}\Phi$$

The columns of matrix $\Phi$ are the $p$ retained mode shapes:

$$\Phi = [\phi_1 \quad \phi_2 \quad \ldots \quad \phi_p]$$

## 2.2. First-order Form of Equations

The second-order modal equations may be rewritten in a first-order form as

$$\begin{aligned} \dot{x}_s &= A_s x_s + B_s u + B_w w + B_s d \\ y &= C_s x_s \\ y^{pr} &= C_1^{pr} x_s + C_2^{pr} \dot{x}_s. \end{aligned} \tag{1}$$

The vector $x_s$ is the plant state vector whose components are

$$x_s = \begin{Bmatrix} q_1 \\ \dot{q}_1 \\ q_2 \\ \dot{q}_2 \\ \vdots \\ \vdots \\ q_p \\ \dot{q}_p \end{Bmatrix}, \tag{2}$$

and the vectors $y$ and $y_{pr}$ are the same plant measurement and performance output vectors as before. The matrix $A_s$ is the plant state matrix and has the form

$$A_s = \begin{bmatrix} A_s^1 & 0 & \cdots & 0 \\ 0 & A_s^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_s^p \end{bmatrix} \tag{3}$$

where

$$A_s^i = \begin{bmatrix} 0 & 1 \\ -\omega_i^2 & -2\zeta_i\omega_i \end{bmatrix}. \tag{4}$$

The matrix $B_s$ is the control input influence matrix, formed by setting its odd-numbered rows to zeros and using the rows of $\bar{H}$ for its even-numbered rows:

$$B_s = \begin{bmatrix} 0 & 0 & \cdots & \cdots & 0 \\ \bar{H}_{11} & \bar{H}_{12} & \cdots & \cdots & \bar{H}_{1m} \\ 0 & 0 & \cdots & \cdots & 0 \\ \bar{H}_{21} & \bar{H}_{22} & \cdots & \cdots & \bar{H}_{2m} \\ \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \cdots & 0 \\ \bar{H}_{p1} & \bar{H}_{p2} & \cdots & \cdots & \bar{H}_{pm} \end{bmatrix} \tag{5}$$

in which $\bar{H}_{ij}$, for example, represents the $(i, j)$ element of matrix $\bar{H}$. The matrix $B_w$ is formed from $\bar{H}_w$ in the same manner.

The measurement output influence matrix, $C_s$ is defined by setting the odd-numbered columns of $C_s$ to the columns of $\bar{C}_p$ and the even numbered columns of $C_s$ to the columns of $\bar{C}_r$:

$$C_s = \begin{bmatrix} \bar{C}_p(1,1) & \bar{C}_r(1,1) & \bar{C}_p(1,2) & \bar{C}_r(1,2) & \cdots & \cdots & \bar{C}_p(1,p) & \bar{C}_r(1,p) \\ \bar{C}_p(2,1) & \bar{C}_r(2,1) & \bar{C}_p(2,2) & \bar{C}_r(2,2) & \cdots & \cdots & \bar{C}_p(2,p) & \bar{C}_r(2,p) \\ \vdots & \vdots & \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & & \ddots & \vdots & \vdots \\ \bar{C}_p(f,1) & \bar{C}_r(f,1) & \bar{C}_p(f,2) & \bar{C}_r(f,2) & \cdots & \cdots & \bar{C}_p(f,p) & \bar{C}_r(f,p) \end{bmatrix} \tag{6}$$

Here, $\bar{C}_p(i, j)$ and $\bar{C}_r(i, j)$ denote the $(i, j)$ element of matrix $\bar{C}_p$ and $\bar{C}_r$, respectively. $C_1^{pr}$ is defined from $\bar{C}_p^{pr}$ and $\bar{C}_r^{pr}$ in the same fashion. $C_2^{pr}$ is defined by setting the odd numbered columns of $C_2^{pr}$ to zeros and the even numbered columns of $C_2^{pr}$ to the columns of $\bar{C}_a^{pr}$.

By substituting the first equation of (1) into the third, the acceleration term can be replaced by feedthrough:

$$\begin{aligned} \dot{x}_s &= A_s x_s + B_s u + B_w w + B_s d \\ y &= C_s x_s \\ y^{pr} &= C^{pr} x_s + D^{pr} u + D_w^{pr} w + D^{pr} d \end{aligned} \tag{7}$$

The performance output influence matrix is given by

$$C^{pr} = C_1^{pr} + C_2^{pr} A_s,$$

while the performance feedthrough matrices are

$$D^{pr} = C_2^{pr} B_s \quad ; \quad D_w^{pr} = C_2^{pr} B_w.$$

Notice that if there is no performance acceleration output ($C_a^{pr} = 0$), then $\bar{C}_a^{pr} = 0$ and $C_2^{pr} = 0$, so both feedthrough matrices, $D^{pr}$ and $D_w^{pr}$, are zero.

## 2.3. Control System Equations

In this paper, it is assumed that the structure is controlled by a linear time-invariant control system. The model of a linear time-invariant control system for a typical flexible structure may be written as

$$\begin{aligned} \dot{x}_c &= A_c x_c + B_c e \\ u &= C_c x_c \\ e &= r - y - v \end{aligned} \tag{8}$$

where $x_c$ denotes the $k \times 1$ vector of control system states; $e$ denotes the $f \times 1$ tracking error vector; $r$ denotes the $f \times 1$ reference input vector; $v$ is the $f \times 1$ measurement noise vector; $A_c$, $B_c$, and $C_c$ represent the $k \times k$ control system state matrix, the $k \times f$ input influence matrix, and the $m \times k$ output influence matrix, respectively; and $y$ is the measurement output vector which was defined in the previous section. Note that the control system equations, as represented by Eq. (8), do not include any feedthrough terms. However, if there are feedthrough terms present in the control system, then augmented dynamics (with roll-off filters) are used to reduce the control system equations to the form given in Eq. (8). This procedure is described in [4]. If the system is open-loop, then $v$ is zero and $y$ is not fed back; so $e$ is $r$, and the control system model becomes:

$$\begin{aligned} \dot{x}_c &= A_c x_c + B_c r \\ u &= C_c x_c \end{aligned} \tag{9}$$

## 2.4. Equations for Open-Loop Configuration

The block diagram of the system for the open-loop configuration is shown in Figure 1.

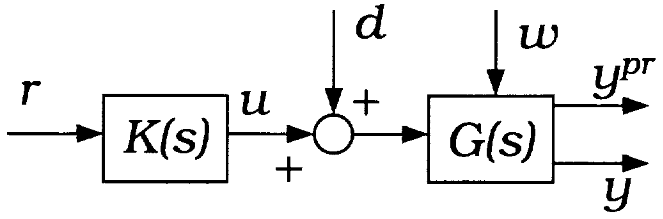

Figure 1. Block diagram for open-loop configuration.

Here, $K(s)$ denotes the controller transfer function matrix, which is defined for all complex $s$ not in the spectrum of $A_c$ as

$$K(s) = C_c(sI - A_c)^{-1}B_c \tag{10}$$

The plant transfer function matrix $G(s)$ is partitioned according to inputs, $d$ and $w$, and outputs, $y^{pr}$ and $y$, and is defined for all complex $s$ not in the spectrum of $A_s$ as

$$G(s) = \begin{bmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \end{bmatrix} \tag{11}$$

with

$$\begin{aligned} G_{11} &= C^{pr}(sI - A_s)^{-1}B_s + D^{pr} \\ G_{12} &= C^{pr}(sI - A_s)^{-1}B_w + D^{pr}_w \\ G_{21} &= C_s(sI - A_s)^{-1}B_s \\ G_{22} &= C_s(sI - A_s)^{-1}B_w \end{aligned} \tag{12}$$

The state-space models of the controller and plant, given by Eqs. (7) and (9), are combined to give

$$\begin{Bmatrix} \dot{x}_s \\ \dot{x}_c \end{Bmatrix} = \begin{bmatrix} A_s & B_sC_c \\ 0 & A_c \end{bmatrix} \begin{Bmatrix} x_s \\ x_c \end{Bmatrix} + \begin{bmatrix} 0 & B_s & B_w \\ B_c & 0 & 0 \end{bmatrix} \begin{Bmatrix} r \\ d \\ w \end{Bmatrix} \tag{13}$$

$$y = [C_s \quad 0] \begin{Bmatrix} x_s \\ x_c \end{Bmatrix} \tag{14}$$

$$y^{pr} = [C^{pr} \quad D^{pr}C_c] \begin{Bmatrix} x_s \\ x_c \end{Bmatrix} + [0 \quad D^{pr} \quad D^{pr}_w] \begin{Bmatrix} r \\ d \\ w \end{Bmatrix} \tag{15}$$

$$u = [0 \quad C_c] \begin{Bmatrix} x_s \\ x_c \end{Bmatrix} \tag{16}$$

For the open-loop system model in Eqs. (13)-(16), there are three possible inputs to the system $(r, d, w)$ and three possible outputs $(y, y^{pr}, u)$. Therefore, a total of nine transfer functions can be defined for the system, with each transfer function corresponding to an input/output pair. Irrespective of the choice of the input/output pair, the computation of open-loop transfer functions from Eqs. (13)-(16) would involve the inverse of matrix $E(s) \equiv \left(sI - \tilde{A}\right)$ at each frequency point $s = j\omega$, where $\tilde{A}$ is the system matrix of the combined plant model and the controller, given by

$$\begin{bmatrix} A_s & B_s C_c \\ 0 & A_c \end{bmatrix}$$

Partition $E(s)$ conformally:

$$E(s) \equiv \begin{bmatrix} E_{11}(s) & E_{12} \\ 0 & E_{22}(s) \end{bmatrix} = \begin{bmatrix} sI - A_s & -B_s C_c \\ 0 & sI - A_c \end{bmatrix} \tag{17}$$

Set $X(s) = E(s)^{-1}$ and partition conformally:

$$X(s) \equiv \begin{bmatrix} X_{11}(s) & X_{12}(s) \\ X_{21}(s) & X_{22}(s) \end{bmatrix} = \begin{bmatrix} E_{11}(s) & E_{12} \\ 0 & E_{22}(s) \end{bmatrix}^{-1} \tag{18}$$

Then, from matrix inversion lemma, one has

$$\begin{aligned} X_{11}(s) &= E_{11}^{-1}(s) \\ X_{12}(s) &= -E_{11}^{-1}(s) E_{12} E_{22}^{-1}(s) \\ X_{21}(s) &= 0 \\ X_{22}(s) &= E_{22}^{-1}(s) \end{aligned} \tag{19}$$

## 2.5. Open-Loop Transfer Functions

Formulae for the nine open-loop transfer functions are derived in this section. Each transfer function formula is stated either in terms of the plant and controller system matrices and terms derived from them in the previous section, or as a simple alteration to a previously derived transfer function. The next section will address issues of computational efficiency.

Subscripts are used in the following to relate input connections to output connections. The first subscript indicates the output and the second subscript indicates the input.

- The transfer function from the tracking command, $r$, to measurement output, $y$, is given by:

$$\begin{aligned} T_{yr}(s) &= G_{21}(s)K(s) = \begin{bmatrix} C_s & 0 \end{bmatrix} \begin{bmatrix} X_{11}(s) & X_{12}(s) \\ X_{21}(s) & X_{22}(s) \end{bmatrix} \begin{bmatrix} 0 \\ B_c \end{bmatrix} \\ &= C_s E_{11}^{-1}(s) B_s C_c E_{22}^{-1}(s) B_c \end{aligned} \tag{20}$$

- The transfer function from the input noise/disturbances, $d$, to measurement output, $y$, is given by:

$$\begin{aligned} T_{yd}(s) &= G_{21}(s) = \begin{bmatrix} C_s & 0 \end{bmatrix} \begin{bmatrix} X_{11}(s) & X_{12}(s) \\ X_{21}(s) & X_{22}(s) \end{bmatrix} \begin{bmatrix} B_s \\ 0 \end{bmatrix} \\ &= C_s E_{11}^{-1}(s) B_s \end{aligned} \tag{21}$$

- The transfer function from the exogenous disturbances, $w$, to measurement output, $y$, is given by:

$$T_{yw}(s) = G_{22}(s) = C_s E_{11}^{-1}(s) B_w \tag{22}$$

- The transfer function from the tracking command, $r$, to performance output, $y^{ps}$, is given by:

$$\begin{aligned} T_{y^{pr}r}(s) &= G_{11}(s)K(s) = \begin{bmatrix} C^{pr} & D^{pr}C_c \end{bmatrix} \begin{bmatrix} X_{11}(s) & X_{12}(s) \\ X_{21}(s) & X_{22}(s) \end{bmatrix} \begin{bmatrix} 0 \\ B_c \end{bmatrix} \\ &= C^{pr} X_{12}(s) B_c + D^{pr} C_c X_{22}(s) B_c \\ &= -C^{pr} E_{11}^{-1}(s) E_{12} E_{22}^{-1}(s) B_c + D^{pr} C_c E_{22}^{-1}(s) B_c \\ &= C^{pr} E_{11}^{-1}(s) B_s C_c E_{22}^{-1}(s) B_c + D^{pr} C_c E_{22}^{-1}(s) B_c \end{aligned} \tag{23}$$

- The transfer function from the input noise/disturbances, $d$, to performance output, $y^{pr}$, is given by:

$$\begin{aligned} T_{y^{pr}d}(s) &= G_{11}(s) = \begin{bmatrix} C^{pr} & D^{pr}C_c \end{bmatrix} \begin{bmatrix} X_{11}(s) & X_{12}(s) \\ X_{21}(s) & X_{22}(s) \end{bmatrix} \begin{bmatrix} B_s \\ 0 \end{bmatrix} + D^{pr} \\ &= C^{pr} X_{11}(s) B_s + D^{pr} \\ &= C^{pr} E_{11}^{-1}(s) B_s + D^{pr} \end{aligned} \tag{24}$$

- The transfer function from the exogenous disturbances, $w$, to performance output, $y^{pr}$, is similarly given by:

$$T_{y^{pr}w}(s) = G_{12}(s) = C^{pr}E_{11}^{-1}(s)B_w + D_w^{pr} \tag{25}$$

- The transfer function from the tracking command, $r$, to control input, $u$, is given by:

$$T_{ur}(s) = K(s) = C_c E_{22}^{-1}(s)B_c \tag{26}$$

- Because the configuration is open-loop the transfer functions from the input noise/disturbances, $d$, and , the exogenous disturbances, $w$, to control input, $u$, are given by $T_{ud} = 0$ and $T_{uw} = 0$.

Now, using one of the equations (20)–(26) to calculate one of these transfer functions, the frequency response function matrix of the open-loop system is evaluated for various values of $s = j\omega$, with $\omega$ taking on the user-specified frequency values. The open-loop gain and phase plots (Bode plots) may then be computed directly from the frequency response function matrix, if desired.

## 2.6. Efficient Calculation of Open-Loop Transfer Functions

If these formulae are used to calculate transfer functions for a large order flexible system, with values of $p$ in the hundreds or thousands, then the presence of the expression $E_{11}^{-1}(s)$ effectively precludes the use of full matrix techniques, both for reasons of computation time and accuracy. Fortunately, the matrix $E_{11}(s) = sI - A_s$ is block diagonal with $2 \times 2$ block diagonal elements. It is a common engineering practice to exploit this sparsity structure for computational efficiency.

It is noted that the expression $E_{11}^{-1}(s)$ always occurs in one of the combinations $E_{11}^{-1}(s)B_s$ or $E_{11}^{-1}(s)B_w$. Additional computational efficiency comes from exploiting the fact that the odd numbered rows of matrices $B_s$ and $B_w$ are zero (see equation (5) and the following remarks). The following illustrates how the sparsity is exploited.

Assume $s$ is not in the spectrum of $A_s$. From equations (3) and (4), it follows that $(sI - A_s)^{-1}$ is block diagonal with the *i-th* block being

$$\left(sI - A_s^i\right)^{-1} = \frac{1}{s^2 + 2\zeta_i\omega_i s + \omega_i^2}\begin{bmatrix} s + 2\zeta_i\omega_i & 1 \\ -\omega_i^2 & s \end{bmatrix}; \quad i = 1, 2, \ldots, p. \tag{27}$$

If the row $i$ of $B_s$ is denoted by $b_s^{(i)}$, if $Q(s)$ is used to represent $(sI - A_s)^{-1}B_s$, and if $Q(s)$ is partitioned as

$$Q(s) = \begin{bmatrix} Q_1(s) \\ Q_2(s) \\ \vdots \\ Q_p(s) \end{bmatrix} \tag{28}$$

where each partition matrix $Q_i(s)$ is a $2 \times m$ matrix, then $Q$ is calculated using the formula

$$Q_i(s) = \left\{1/\left(s^2 + 2\zeta_i\omega_i s + \omega_i^2\right)\right\} \begin{bmatrix} b_s^{(2i)} \\ sb_s^{(2i)} \end{bmatrix}; \quad i = 1, 2, \ldots, p. \tag{29}$$

A similar approach is followed in the computation of $E_{11}^{-1}(s)B_w$.

The term $E_{22}^{-1}(s) = (sI - A_c)^{-1}$ appears in several of the transfer function formulae, always in the context $C_cE_{22}^{-1}(s)B_c$. The $k \times k$ matrix $sI - A_c$ is generally a full matrix. Consequently, the computation of the controller transfer function $C_cE_{22}^{-1}(s)B_c$ is achieved through conventional approaches (see, e.g., the work of Laub [1, 2]).

Most of the open-loop transfer function formulae contain one of the expressions $C_sE_{11}^{-1}(s)B_s$, $C_sE_{11}^{-1}(s)B_w$, $C^{pr}E_{11}^{-1}(s)B_s$, or $C^{pr}E_{11}^{-1}(s)B_w$. By comparing the FLOP (FLoating point OPeration) count for alternate ways of computing one of these expressions, the computational advantage of utilizing the block diagonal structure of $E_{11}^{-1}(s)$ can be quantified.

Observe that

$$G_{21}(s) = C_sE_{11}^{-1}(s)B_s = C_sQ(s)$$

The standard, full matrix way to calculate $Q(s)$ would involve first performing an LU decomposition of $sI - A_s$, followed by a backward and then forward solution of the triangular systems of equations using the columns of $B_s$ as right-hand sides. The FLOP count for this is $O(p^3) + O(p^2m)$, so $G_{21}(s)$ is computed in $O(p^3) + O(p^2m) + O(pfm)$ FLOPS. Thus, in the typical case where system size is much larger than the number of disturbances, the calculation time per frequency point is a cubic function of system size.

If this calculation must be repeated for many values of $s$ (a typical scenario), the technique of [1, 2] has a better average FLOP count. An initial $O(p^3)$ orthogonal transformation must be done once; then for each $s$, $Q(s)$ is calculated in $O(p^2m)$ FLOPs, so $G_{21}(s)$ is computed in $O(p^2m)+O(pfm)$ FLOPs. Thus, if the number of frequencies for which this calculation must be repeated is large enough (see the discussion in the last paragraph of Appendix A), the $O(p^3)$ start-up cost is distributed over the frequency points such that the calculation time per frequency point is a quadratic function of system size.

When the calculation is done as in equations (28) and (29), the flop count is $O(pm)$, so $G_{21}(s)$ is computed in $O(pfm)$ FLOPs. Thus, the calculation time per frequency point is a linear function of system size. This represents a substantial savings, particularly when a large number of modes is necessary to capture the dynamics of the system.

## 2.7. Equations for Closed-Loop Configuration

The block diagram of the system for the closed-loop configuration is shown in Figure 2.

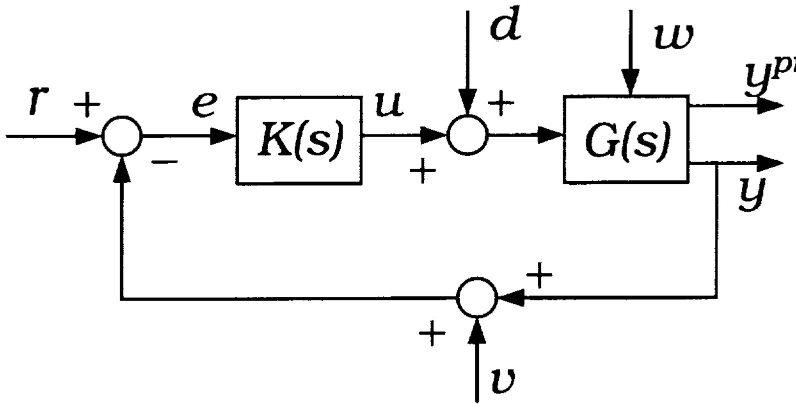

Figure 2. Block diagram for closed-loop configuration.

The closed-loop system is more complicated. Using equations (7) and (8) and defining

$$\tilde{A} \equiv \begin{bmatrix} A_s & B_sC_c \\ -B_cC_s & A_c \end{bmatrix}, \tag{30}$$

the closed-loop dynamics of the controlled structure may be written as:

$$\begin{Bmatrix} \dot{x}_s \\ \dot{x}_c \end{Bmatrix} = \tilde{A} \begin{Bmatrix} x_s \\ x_c \end{Bmatrix} + \begin{bmatrix} 0 & B_s & B_w & 0 \\ B_c & 0 & 0 & -B_c \end{bmatrix} \begin{Bmatrix} r \\ d \\ w \\ v \end{Bmatrix} \tag{31}$$

$$y = [C_s \quad 0] \begin{Bmatrix} x_s \\ x_c \end{Bmatrix} \tag{32}$$

$$y^{pr} = [C^{pr} \quad D^{pr} C_c] \begin{Bmatrix} x_s \\ x_c \end{Bmatrix} + [0 \quad D^{pr} \quad D_w^{pr} \quad 0] \begin{Bmatrix} r \\ d \\ w \\ v \end{Bmatrix} \tag{33}$$

$$e = [-C_s \quad 0] \begin{Bmatrix} x_s \\ x_c \end{Bmatrix} + [I \quad 0 \quad 0 \quad -I] \begin{Bmatrix} r \\ d \\ w \\ v \end{Bmatrix} \tag{34}$$

$$u = [0 \quad C_c] \begin{Bmatrix} x_s \\ x_c \end{Bmatrix} \tag{35}$$

For the closed-loop system model in Eqs. (31)-(35), there are four possible inputs to the system $(r, d, w, v)$ and four possible outputs $(y, y^{pr}, e, u)$. Therefore, a total of 16 transfer functions can be defined for the system, with each transfer function corresponding to an input/output pair.

Irrespective of the choice of the input/output pair, the computation of closed-loop transfer functions from Eqs. (31)-(35) would require the inverse of matrix $sI - \tilde{A}$ at each frequency point $s = j\omega$. Observing the closed-loop state matrix $\tilde{A}$, it is obvious the block triangular form of the open-loop configuration has been destroyed by the coupling generated by the feedback connection of plant and the control system. However, the initial sparsity of the plant state matrix $A_s$ is still intact. This sparsity is exploited to develop an efficient method for the computation of closed-loop frequency response function matrix of the controlled flexible structure. If sparsity is not exploited and many structural modes are modeled ($p$ is large), it follows that a large computational effort would be required to calculate the closed-loop frequency response function matrix, since this would involve the computation of the matrix term $\left(sI - \tilde{A}\right)^{-1} \tilde{B}$, where $sI - \tilde{A}$ is of order $2p+k$. Here, $\widetilde{B}$ represents one of the block columns in equation (31) and the computation must be repeated for each value of $s = j\omega$ for all desired frequency values $\omega$.

In the following it is assumed that $s$ is not in the spectrum of $\widetilde{A}$ (necessary for the transfer function even to be defined) and it is further assumed that $s$ is not in the spectrum of $A_s$. This further assumption is needed to enable some algebraic manipulation, and should not adversely affect the applicability of the following results. On the one hand, since $A_s$ is the plant state matrix for a linear model of a flexible structure, its eigenvalues occur either at 0 (corresponding to rigid body modes) or in the left half plane (corresponding to damped flexible modes). On the other hand, it is anticipated that these results will be used to compute closed-loop transfer functions for $s = j\omega$ with $\omega > 0$. Thus, excluding the eigenvalues of $A_s$ from the domain of applicability of these results does not impact the anticipated usage.

The matrix term $sI - \widetilde{A}$ may be written as

$$sI - \widetilde{A} = \begin{bmatrix} sI - A_s & -B_s C_c \\ B_c C_s & sI - A_c \end{bmatrix} \equiv \begin{bmatrix} E_{11}(s) & E_{12} \\ E_{21} & E_{22}(s) \end{bmatrix}. \qquad (36)$$

Introduce the notation:

$$\begin{bmatrix} X_{11}(s) & X_{12}(s) \\ X_{21}(s) & X_{22}(s) \end{bmatrix} \equiv \left(sI - \widetilde{A}\right)^{-1} = \begin{bmatrix} E_{11}(s) & E_{12} \\ E_{21} & E_{22}(s) \end{bmatrix}^{-1} \qquad (37)$$

The assumptions which have been made about $s$ insure that the inverses in (37) exist, as does $E_{11}^{-1}$. Rewrite (37) as:

$$\begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix} = I \qquad (38)$$

Expanding the lower left block of (38) and solving for $X_{21}$ gives $X_{21} = -X_{22}E_{21}E_{11}^{-1}$. Expanding the lower right block of (38), substituting the previous expression for $X_{21}$, and factoring gives $X_{22}\Delta = I$, where $\Delta = E_{22} - E_{21}E_{11}^{-1}E_{12}$. This demonstrates that $\Delta$ is invertible, and justifies the application of the block matrix inversion formula given in [5, page 898] to find the inverse of the block matrix in Equation (37):

$$\begin{aligned}
\Delta &= E_{22} - E_{21}E_{11}^{-1}E_{12} \\
X_{11} &= E_{11}^{-1} + E_{11}^{-1}E_{12}\Delta^{-1}E_{21}E_{11}^{-1} \\
X_{12} &= -E_{11}^{-1}E_{12}\Delta^{-1} \\
X_{21} &= -\Delta^{-1}E_{21}E_{11}^{-1} \\
X_{22} &= \Delta^{-1}
\end{aligned} \tag{39}$$

## 2.8. Closed-Loop Transfer Functions

Formulae for the sixteen closed-loop transfer functions are derived in this section. Each transfer function formula is stated either in terms of the plant and controller system matrices and terms derived from them in the previous section, or as a simple alteration to a previously derived transfer function. The next section will address issues of computational efficiency.

- By combining information from equations (36) and (39), it is seen that the term $\Delta(s)$, which is used in the computation of all of these transfer functions, can be calculated by the formula:

$$\Delta(s) = E_{22}(s) + B_cC_sE_{11}^{-1}(s)B_sC_c \tag{40}$$

- The transfer function from the tracking command, $r$, to measurement output, $y$, is given, with the aid of Eq. (37), by:

$$\begin{aligned}
T_{yr}(s) &= (I + G_{21}(s)K(s))^{-1}G_{21}(s)K(s) \\
&= \begin{bmatrix} C_s & 0 \end{bmatrix} \begin{bmatrix} X_{11}(s) & X_{12}(s) \\ X_{21}(s) & X_{22}(s) \end{bmatrix} \begin{bmatrix} 0 \\ B_c \end{bmatrix} \\
&= C_sX_{12}(s)B_c = -C_sE_{11}^{-1}(s)E_{12}\Delta^{-1}(s)B_c \\
&= C_sE_{11}^{-1}(s)B_sC_c\Delta^{-1}(s)B_c
\end{aligned} \tag{41}$$

- The transfer function from the input noise/disturbance, $d$, to measurement output, $y$, is given by:

$$\begin{aligned}
T_{yd}(s) &= (I + G_{21}(s)K(s))^{-1}G_{21}(s) = \begin{bmatrix} C_s & 0 \end{bmatrix} \begin{bmatrix} X_{11}(s) & X_{12}(s) \\ X_{21}(s) & X_{22}(s) \end{bmatrix} \begin{bmatrix} B_s \\ 0 \end{bmatrix} \\
&= C_sX_{11}(s)B_s \\
&= C_sE_{11}^{-1}(s)B_s - C_sE_{11}^{-1}(s)B_sC_c\Delta^{-1}(s)B_cC_sE_{11}^{-1}(s)B_s
\end{aligned} \tag{42}$$

- The transfer function from the exogenous disturbances, $w$, to measurement output, $y$, is given by:

$$\begin{aligned} T_{yw}(s) &= (I + G_{21}(s)K(s))^{-1}G_{22}(s) = [C_s \quad 0]\begin{bmatrix} X_{11}(s) & X_{12}(s) \\ X_{21}(s) & X_{22}(s) \end{bmatrix}\begin{bmatrix} B_w \\ 0 \end{bmatrix} \\ &= C_s X_{11}(s) B_w \\ &= C_s E_{11}^{-1}(s) B_w - C_s E_{11}^{-1}(s) B_s C_c \Delta^{-1}(s) B_c C_s E_{11}^{-1}(s) B_w \end{aligned} \tag{43}$$

- The transfer function from the measurement noise, $v$, to measurement output, $y$, is given by:

$$T_{yv}(s) = -T_{yr}(s) \tag{44}$$

- The transfer function from the tracking command, $r$, to performance output, $y^{pr}$, is given, with the aid of Eq. (37), by:

$$\begin{aligned} T_{y^{pr}r}(s) &= G_{11}(s)(I + G_{21}(s)K(s))^{-1}K(s) \\ &= [C^{pr} \quad D^{pr}C_c]\begin{bmatrix} X_{11}(s) & X_{12}(s) \\ X_{21}(s) & X_{22}(s) \end{bmatrix}\begin{bmatrix} 0 \\ B_c \end{bmatrix} \\ &= C^{pr}X_{12}(s)B_c + D^{pr}C_c X_{22}(s) B_c \\ &= -C^{pr}E_{11}^{-1}(s)E_{12}\Delta^{-1}(s)B_c + D^{pr}C_c\Delta^{-1}(s)B_c \\ &= C^{pr}E_{11}^{-1}(s)B_s C_c \Delta^{-1}(s)B_c + D^{pr}C_c\Delta^{-1}(s)B_c \end{aligned} \tag{45}$$

- The transfer function from the input noise/disturbance, $d$, to performance output, $y^{pr}$, is given, with the aid of Eq. (37), by:

$$\begin{aligned} T_{y^{pr}d}(s) &= G_{11}(s)(I + K(s)G_{21}(s))^{-1} \\ &= [C^{pr} \quad D^{pr}C_c]\begin{bmatrix} X_{11}(s) & X_{12}(s) \\ X_{21}(s) & X_{22}(s) \end{bmatrix}\begin{bmatrix} B_s \\ 0 \end{bmatrix} + D^{pr} \\ &= C^{pr}X_{11}(s)B_s + D^{pr}C_c X_{21}(s)B_s + D^{pr} \\ &= C^{pr}E_{11}^{-1}(s)B_s - C^{pr}E_{11}^{-1}(s)B_s\Big\{C_c\Delta^{-1}(s)B_c C_s E_{11}^{-1}(s)B_s\Big\} \\ &\quad - D^{pr}\Big\{C_c\Delta^{-1}(s)B_c C_s E_{11}^{-1}(s)B_s\Big\} + D^{pr} \end{aligned} \tag{46}$$

- The transfer function from the exogenous disturbance, $w$, to performance output, $y^{pr}$, is given, with the aid of Eq. (37), by:

$$\begin{aligned} T_{y^{pr}w}(s) &= G_{12}(s)(I+K(s)G_{21}(s))^{-1} \\ &= [C^{pr} \quad D^{pr}C_c]\begin{bmatrix} X_{11}(s) & X_{12}(s) \\ X_{21}(s) & X_{22}(s)\end{bmatrix}\begin{bmatrix} B_w \\ 0 \end{bmatrix} + D_w^{pr} \\ &= C^{pr}E_{11}^{-1}(s)B_w - C^{pr}E_{11}^{-1}(s)B_s\left\{C_c\Delta^{-1}(s)B_cC_sE_{11}^{-1}(s)B_w\right\} \\ &- D^{pr}\left\{C_c\Delta^{-1}(s)B_cC_sE_{11}^{-1}(s)B_w\right\} + D_w^{pr} \end{aligned} \tag{47}$$

- The transfer function from the measurement noise, $v$, to performance output, $y^{pr}$, is given by:

$$T_{y^{pr}v}(s) = -T_{y^{pr}r}(s) \tag{48}$$

- The transfer function from the reference command, $r$, to reference error is given by:

$$T_{er}(s) = I - T_{yr}(s) \tag{49}$$

- The transfer function from the input noise, $d$, to reference error, $e$, is given by:

$$T_{ed}(s) = -T_{yd}(s) \tag{50}$$

- The transfer function from the exogenous disturbances, $w$, to reference error, $e$, is given by:

$$T_{ew}(s) = -T_{yw}(s) \tag{51}$$

- The transfer function from the measurement noise, $v$, to reference error, $e$, is given by:

$$T_{ev}(s) = T_{yr}(s) - I \tag{52}$$

- The transfer function from the tracking command, $r$, to control input, $u$, is given, with the aid of Eq. (37), by:

$$\begin{aligned} T_{ur}(s) &= K(s)(I+G_{21}(s)K(s))^{-1} = [0 \quad C_c]\begin{bmatrix} X_{11}(s) & X_{12}(s) \\ X_{21}(s) & X_{22}(s)\end{bmatrix}\begin{bmatrix} 0 \\ B_c \end{bmatrix} \\ &= C_cX_{22}(s)B_c = C_c\Delta^{-1}(s)B_c \end{aligned} \tag{53}$$

- The transfer function from the input noise/disturbance, $d$, to control input, $u$, is given, with the aid of Eq. (37), by:

$$T_{ud}(s) = -K(s)(I + G_{21}(s)K(s))^{-1}G_{21}(s) = [0 \quad C_c]\begin{bmatrix} X_{11}(s) & X_{12}(s) \\ X_{21}(s) & X_{22}(s) \end{bmatrix}\begin{bmatrix} B_s \\ 0 \end{bmatrix}$$
$$= -C_c X_{21}(s)B_s = -C_c\Delta^{-1}(s)B_cC_sE_{11}^{-1}(s)B_s \tag{54}$$

- The transfer function from the exogenous disturbances, $w$, to control input, $u$, is given, with the aid of Eq. (37), by:

$$\begin{aligned} T_{uw}(s) &= -K(s)(I + G_{21}(s)K(s))^{-1}G_{22}(s) \\ &= [0 \quad C_c]\begin{bmatrix} X_{11}(s) & X_{12}(s) \\ X_{21}(s) & X_{22}(s) \end{bmatrix}\begin{bmatrix} B_w \\ 0 \end{bmatrix} \\ &= -C_c X_{21}(s)B_w = -C_c\Delta^{-1}(s)B_cC_sE_{11}^{-1}(s)B_w \end{aligned} \tag{55}$$

- The transfer function from the measurement noise, $v$, to control input, $u$, is given by:

$$T_{uv}(s) = -T_{ur}(s) \tag{56}$$

Now, using one of the equations (41)–(56) to calculate one of these transfer functions, the frequency response function matrix of the closed-loop system is evaluated for various values of $s = j\omega$, with $\omega$ taking on the user-specified frequency values. The closed-loop gain and phase plots (Bode plots) may then be computed directly from the frequency response function matrix, if desired.

## 2.9. Efficient Calculation of Closed-Loop Transfer Functions

With the transfer functions written in this form, the following computational efficiencies are observed:

□ Since $E_{11} = sI - A_s$, and $B_s$ and $B_w$ have zeros in the odd numbered rows, the terms $E_{11}^{-1}B_s$ and $E_{11}^{-1}B_w$ can be computed using the techniques presented for efficient computation of the open-loop transfer function (see the §2.6 through equation (29)). Achieving this efficiency was the entire point of the algebraic manipulations done in achieving the forms of the transfer function formulae given in the previous sections.

□ The computation of $\Delta^{-1}B_c$ is done as a full matrix computation (a possible improvement to this is presented in Appendix A), but since $\Delta$ is of the same order as the control system, which is usually small compared to the order of the analysis model of the plant, it should not be very costly to compute. Following accepted contemporary numerical analytic practice, $\Delta^{-1}B_c$ should be computed by solving systems of linear equations using $\Delta$ as the coefficient matrix and the columns of $B_c$ as right hand sides ([6, Chapter 3]).

- ☐ The expected shapes of the matrices and the exploitation of common sub-expressions make it advisable not to precompute some of the matrix products in equations (40)–(56) which are independent of the frequency parameter $s$ (if $X$ is a tall, skinny matrix, and $Y$ is a short, wide matrix, so that $W = XY$ is both tall and wide; and if $z$ is a vector, then calculating $X(Yz)$ is cheaper than calculating $Wz$).
- ☐ Common sub-expressions, such as those mentioned in the previous items and those enclosed in braces in some of the transfer function equations are computed once per frequency, saved, and reused. Some of the intermediate results formed in computing $\Delta$ can be reused in many of the transfer function formulae.
- ☐ If there are no acceleration sensors in the performance outputs, so that the feedforward matrices $D^{pr}$ and $D^{pr}_w$ are null, the software implementations of these computations may bypass any terms in the transfer function formulae which contain either of these matrices.

The closed-loop system matrix $\tilde{A}$ (equation (30)) has order $2p + k$. As in the discussion of the open-loop calculation, if $T_{y^{pr}w}(s)$ is calculated as

$$T_{y^{pr}w}(s) = [C^{pr} \quad D^{pr}C_c]\left(sI - \tilde{A}\right)^{-1}\begin{bmatrix} B_w \\ 0 \end{bmatrix} + D^{pr}_w$$

using standard full matrix techniques, the computation takes $O\left((2p+k)^3\right) + O\left((2p+k)^2 g\right) + O((2p+k)lg)$ FLOPs per frequency point. Using the technique of [1] and [2], if the number of frequency points for which the calculation is to be repeated is on the order of $2p + k$ or more, then $T_{y^{pr}w}(s)$ can be calculated in $O\left((2p+k)^2 g\right) + O((2p+k)lg)$ FLOPs per frequency point. Again, these are cubic and quadratic functions, respectively, of the system size. By counting FLOPs resulting from subroutine calls and DO loops in the FORTRAN software used to implement the calculation of equation (47), it is determined that $T_{y^{pr}w}(s)$ can be calculated in

$$O(p(lg + gf + fm + gm) + k^3 + k^2(g + m) + k(gf + fm + gm) + lgm) \qquad (57)$$

FLOPs per frequency point. Again, this is a linear function of the number of retained modes $p$.

In order to evaluate the significance of Eq. (57), it is useful to recall the meanings of the variables in it, and to take note of their expected relative sizes in the applications to which this theory is intended to be applied. The number of retained modes $p$ could be the largest by far of these variables, with values ranging even into the thousands. The number of performance outputs $l$ and the number of controller states $k$ are of moderate size, perhaps as large as multiples of ten. The number of disturbance inputs $g$ is much smaller, perhaps as large as 10 or so, while the number of measurement outputs $f$ and the number of control inputs to the plant $m$ are even smaller, typically being in the low single digits. Therefore, the terms involving $p$ should be looked on as the dominant part of Eq. (57). Of the remaining terms, the $O(k^3)$ term is the next most dominant.

If a FLOP count expression were to be developed for another of the transfer function formulae, it would differ in some details. However, two important points would remain the same: $p$ would appear linearly, and, of the terms which do not depend on $p$, the $O(k^3)$ term would continue to appear and dominate.

It is therefore interesting to note that, if the number of frequency points at which the transfer function is to be evaluated is large enough, the term $O(k^3)$ in this last expression, which comes from performing a LU decomposition on the matrix $\Delta$, can be reduced enough to become part of the terms which are of lower order in $k$. This reduction is based on applying the technique of [1, 2] to $E_{22}$ and making use of the observation that the other term in the definition of $\Delta$ is, in the expected applications, of low rank. This is seen by observing from Eq. (40) that the rank of the term added to $E_{22}$ to get $\Delta$ is at most $\min(f, m)$. Details of this alternative calculation are given in Appendix A.

## 2.10. Discrete Systems

The computational procedure outlined for the calculation of open-loop or closed-loop transfer functions of the system extends to the case wherein the plant and the controller are represented by linear discrete-time systems with little adjustment. These adjustments and considerations are:

1. The plant state and influence matrices, $A_s, B_s, B_w, C_s, C^{pr}, D^{pr}$, and $D_w^{pr}$, which have been used in the computation of open-loop or closed-loop trans-

fer functions, take their discrete-time forms (instead of their continuous-time forms).

2. The matrix $E_{11} = sI - A_s$ continues to keep its block diagonal form with 2x2 block diagonals. However, the special form of Eq. (27) for the inverse of the 2x2 block diagonal elements does not hold anymore. Instead, the general inverse for a 2x2 matrix is used.
3. The influence matrices $B_s$ and $B_w$ do not possess the property of having zeros in the odd numbered rows; they are full matrices with no inherent sparsity. This means that, in the computation corresponding to Eq. (29), both an odd numbered and an even numbered row of the "B" matrix are involved in the computation, as opposed to only an even numbered row for the continuous-time case.
4. The frequency points $s$ which are used in the computations are computed from

$$s = e^{j\omega T_s}$$

where $\omega$ are the desired frequencies in rad/s and $T_s$ denotes the sampling frequency of the discrete-time system.

## 2.11. Software Implementation

The algorithms developed in this paper has evolved over time. In the initial work as reported in [3], only the formulae for the open-loop and closed-loop transfer functions for performance output as a function of disturbance input, called $T_{y^{pr}w}$ in this paper, were developed; and these only for the continuous case. Software capabilities also evolved during this time. The initial implementation of software to evaluate $T_{y^{pr}w}$ is distinctly different from the current implementation. The techniques suggested in Appendix A have not yet been tried out in software.

### 2.11.1. Initial Implementation

The evaluations of the open- and closed-loop transfer function $T_{y^{pr}w}$ were originally implemented using MATLAB function M-files (MATLAB, a product of The MathWorks, Inc., is "a technical computing environment for high-performance numeric computation and visualization" [7, page *i*]), and as FORTRAN 77 code which is then accessed through MATLAB using the MEX-file external interface facility. The M-files have the advantage of being relatively easy to write and of being portable to any computer capable of running MATLAB. The FORTRAN MEX-files are more labor intensive to program, but for those computer systems which are capable of compiling them, they perform the calculations faster. This is illustrated in §3.

The M-files contain straightforward implementations of the calculations presented in the open- and closed-loop formulae for $T_{y^{pr}w}$. The FORTRAN source code for the MEX-files uses the Basic Linear Algebra Subprograms (BLAS, [8–14]) to perform vector-vector, vector-matrix, and matrix-matrix operations. In addition, LAPACK ([15]) subroutine ZGESV, a complex double precision linear equation solver, is used to calculate $\Delta^{-1}\left(B_c C_s E_{11}^{-1}(s) B_w\right)$. It is in this form that the transfer function evaluations are implemented in version 1 of the software package PLATSIM [4], which is available from COSMIC (LAR-15287).

### 2.11.2. Current Implementation

Implementing the full scope of the computational formulae developed in this paper was a task of significantly greater magnitude than was involved in the initial implementation. In the continuous case, there were seven nontrivial open-loop transfer functions and 16 closed-loop transfer functions for a total of 23. Extending the work to the discrete domain added another 23 transfer functions. Fortunately, there were some similarities in the forms of the calculations between some of the transfer functions.

Exploiting these similarities in form, it is possible to calculate all seven continuous open-loop transfer functions using just four function M-files, one executive routine and three which do the actual calculations. Similarly, the 16 continuous closed-loop transfer functions can be computed using just six M-files, one executive and five performing the actual calculations. The discrete case uses the same software structure, so any of the 46 distinct transfer functions can be calculated using a suite of 20 function M-files.

It would have been a programming task of nontrivial magnitude to render these 46 transfer function formulae in FORTRAN or C code from which MEX-files could be compiled. Fortunately, by the time this theory had been developed, The MathWorks, Inc. had developed the MATLAB compiler [16]. This is a utility which translates function M-files into C code which can either be used in stand-alone applications (with the addition of the MATLAB C math library) or be compiled into MEX-files.

There are some limitations as to which MATLAB commands can be compiled. Also, some calculations can be programmed in MATLAB in more than one way, such that one program is more efficient as an M-file and another is more efficient as compiled code. So, slightly different versions of the 20 M-files are maintained for compilation into MEX-files as opposed to direct execution by MATLAB. These 20 M-files are used to compile four MEX-files; one to perform the computation of all of the open-loop transfer functions in the continuous time case, one for closed-loop continuous time transfer functions, and one each for open-loop and closed-loop discrete time transfer functions. It is in this form that the transfer function evaluations are implemented in version 2 of the software package PLATSIM [17].

## 3. Numerical Examples

A number of numerical examples are presented to demonstrate the efficiency and accuracy of the algorithm presented in this paper. Execution time comparisons are made to two standard full matrix methods of calculating the frequency response function of a closed-loop system and a block elimination method for linear systems with a multi-row and column boarder. Accuracy assessments are made by comparing these answers amongst themselves and with answers calculated using quadruple precision (128 bit) floating point computer arithmetic. Except where specifically noted otherwise, all software used in this study utilized ANSII standard double precision (64 bit) floating point computer arithmetic. All tests were performed on a Sun SPARC 10 workstation.

The data come from a model for the EOS-AM-1 spacecraft used in a jitter reduction study ([18]). This spacecraft (see Fig. 3) has a diameter of 3.5 m, a length of 6.8 m, and a weight of 5190 kg.

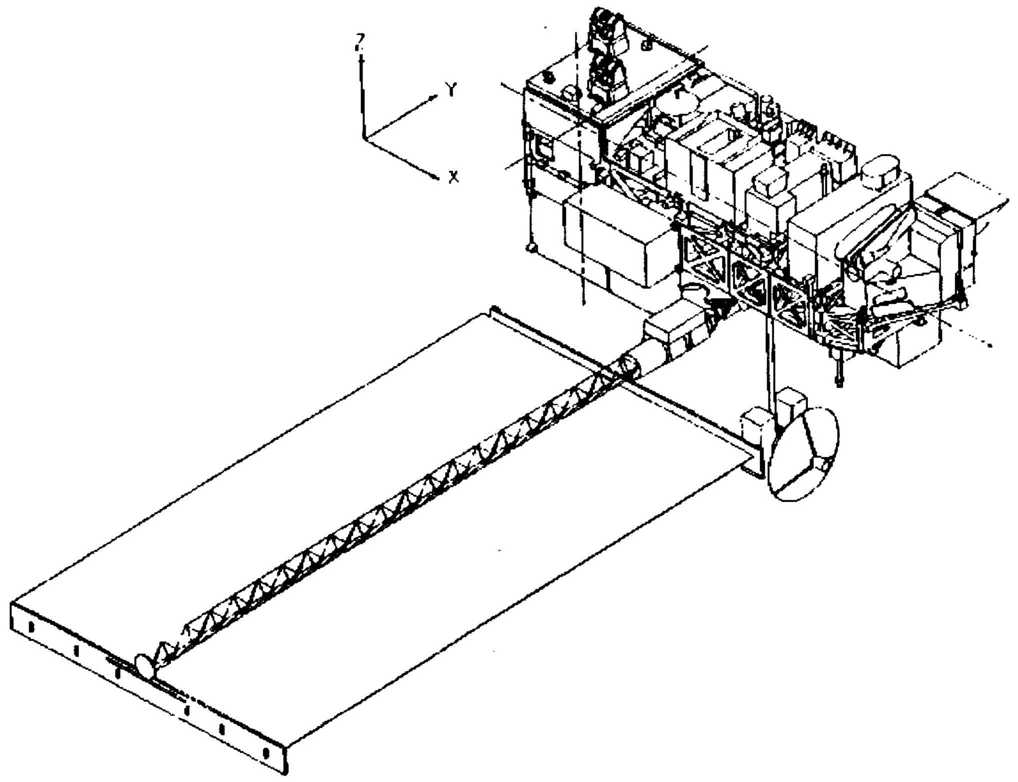

Figure 3. Schematic diagram of EOS-AM-1 spacecraft.

The structural model contains 703 modes for a potential 1406 plant states. This modal model includes all modes with frequencies less than or equal to 200 Hz. The reason for using such a large flexible model of the spacecraft is to capture the diverse dynamic characteristics of the disturbances that act on it. The frequency of disturbances can vary from orbital frequency, such as gravity gradient, to sub-Hertz frequency, such as mass imbalance on scanning mirrors, to frequencies of more than 100 Hz, such as are generated by the cryocoolers.

There are 6 rigid body modes and 697 flexible modes ranging from 1.24 to 1564 radians per second. The 6 measurement outputs are the spacecraft's roll, pitch, and yaw attitudes, and their rates, at the spacecraft navigational unit. The three actuators are x-, y-, and z-axis torquers. The control system has 39 states. Depending on which of the four types of input and four types of output were being used, tests were run with up to 10 channels of input and up to 27 channels of output. There is enough similarity among the formulae for the open- and closed-loop cases and between the continuous and discrete time cases that it was judged sufficient to conduct this testing for the closed-loop continuous time case.

## 3.1. Test 1: Initial Implementation

In this test, the closed-loop continuous time transfer function $T_{y^{pr}w}$ is calculated using the initial implementation of the software. Several parameters are varied in order to assess their effect on execution time of the software for the various algorithms.

Cases were run using position measurements at each output, resulting in no feedforward term, and using acceleration measurements at each output, resulting in a feedforward term being present.

All algorithms used in this timing study are intended to be used to calculate the frequency response matrix at multiple frequency points, so that the frequency response may be plotted (as, e.g., Bode plots). They all have some calculations which are done once per entry into the algorithm and other calculations which are done once for every frequency point. To take this into account, all cases were run over a range of 200 frequency points and most of them were rerun using 2000 points (the exceptions were the cases which would have required more than 5 days of cpu time to complete). In all cases, the points were logarithmically distributed between frequencies of .01 and 10000 radians per second.

The executions times of test codes are compared on 12 representative problems. Three different plant sizes were used: a small plant with 1 input, 1 output, and 61 states (24 structural and 39 controller states); a medium plant with 3 inputs, 5 outputs, and 221 states (184 structural and 39 controller states); and a large plant with 10 inputs, 27 outputs, and 1445 states (1406 structural and 39 controller states). For each plant size, two different sets of outputs were used: one with no feedforward term (i.e., no acceleration outputs were used as performance outputs) and one with feedforward. The frequency response function of each of these 6 plants was calculated at a short (200 values) vector of frequency values and at a long (2000 values) vector of frequency values.

### 3.1.1. Software Used in Test 1

In Test 1, two software realizations of the closed-loop frequency response function algorithm discussed earlier are compared to two software realizations of previously available algorithms for the calculation of transfer functions.

The algorithm presented in this paper is programmed both as a MATLAB function M-file and as FORTRAN 77 code which is then accessed through MATLAB using the MEX-file external interface facility. These will be called, respectively, the new M-code and the new MEX-code.

One of the programs used for comparison makes use of the algorithm in [1] and [2]. The FORTRAN code in [2] is in single precision; Laub's own double precision FORTRAN code is imbedded in the software package FREQ ([19]) and was used here. This test code is purely in FORTRAN 77. It will be called the old FORTRAN code.

Preliminary testing indicated that, in order to get a reasonably well-conditioned matrix for the $sI - \tilde{A}$ expression in the Laub code, it was necessary to exercise the built-in option of balancing the $\tilde{A}$ matrix. The unbalanced matrix was particularly ill-conditioned at low frequencies. This can be attributed to the presence of the rigid body (0 frequency) modes. In the Laub code, balancing was coupled with the extraction of the eigenvalues of $\tilde{A}$. As Laub wrote this code, the same value of the input flag which signaled the code to balance the $\tilde{A}$ matrix also signaled the code to extract its eigenvalues. For purposes of timing tests here, the Laub code was modified so that the portion which extracts eigenvalues was bypassed.

A second program used for comparison is the MathWorks M-file `freqrc.m`, an undocumented utility routine in the *Robust Control Toolbox*, [20], which calculates (to quote the on-line program preamble comments) "Continuous complex frequency response (MIMO)". This will be called the old M-code. Once again, to achieve reasonable accuracy, it was necessary to balance $\tilde{A}$. This was done using MATLAB built-in routine `balance`.

### 3.1.2. Timing Comparisons

Particularly on the larger plants, the new algorithm performs dramatically faster than the older programs. This should not be taken as an indictment of the older algorithms. The older algorithms were designed for arbitrary plants while the new takes full advantage of the particular pattern of sparsity which results from using the modal model of a flexible structure. On the other hand, when the new algorithm is applicable, it enables analysis of structures of much larger order than would be practical or even possible before.

Table 1 gives the time in seconds to calculate the frequency response function using each of the 4 test routines for each of these 12 cases (except that the old M-code does not attempt the two largest cases).

One conclusion to be drawn from this table is that the timing values returned by the system timing software are not totally consistent with each other. The computations by first three software packages include an up front check to see if feedforward is present. The bulk of the code is executed whether feedforward is present or not. If feedforward is present, additional code is executed which should take additional time. But in 9 of 18 cases, the table shows the feedforward case taking less time than the one without.

That said, there are still significant trends to be observed in this timing data. The new MEX-code is significantly faster than the new M-code; in the more important larger cases, about 3 times as fast. This justifies the effort of rendering the algorithm in FORTRAN and writing the interface necessary to access it through MATLAB.

Comparing the new MEX-code, which is FORTRAN based, with the old FORTRAN code shows that for the small system, the old code more than holds its own. This is not unexpected, since in the small system, the controller dominates the count of states. Thus, there is relatively little sparsity from the structural part of the plant of which the new MEX-code may take advantage. But even in the medium size case, the new code is 4 to 7 times faster than the old. This is getting near the size at which conventional numerical analytic wisdom would place the limits of applicability of the old, full matrix based, technique. For large cases, the new MEX-code is about 40 times faster than the old FORTRAN code. Since the CPU time for the old FORTRAN code is expected to grow quadratically with the number of system states, while that of the new technique is expected to grow only linearly, it is not surprising that the difference between them in the largest example is so great.

In what is called here the *old M-code*, MathWorks actually used M-files for outer loop logic control to drive a built in function, `ltifr`. This function calculates the matrix $G$ whose columns are $(s(i)I - A)^{-1}b$ where $s$ is a vector of complex numbers (set in the present application to $j\omega$, where $j^2 = -1$ and $\omega$ is a vector of frequencies), $A$ is a system matrix (set in this application to $\tilde{A}$), and $b$ is a column vector (set in this application to a column of the $B_w$ matrix). The on-line help for `ltifr` states that it "implements, in high speed" what the user could calculate by looping through the elements of $s$ and building $G$ one column at a time. Despite this, it is only competitive on the smallest system, and then only against the new M-code which utilizes MATLAB built-in functions only at the more primitive level of basic matrix operations.

All of the algorithms tested have some "once per entry" calculations in addition to the calculations which occur once per frequency value. Thus, the time for the 2000 point calculations *should* never be more than 10 times that for the 200 point calculations. In Table 1, there are several exceptions to this. This reinforces the previous remark that the numbers returned by the computer system timing routines are, at best, approximate. However, from looking at the largest case, it can be reasonably concluded that the "once per entry" overhead is fairly small in both realizations of the new algorithm while being substantial in the old FORTRAN code, at least for large systems. This is expected, since for a system of order $n$ (all other parameters being held fixed), the "once per entry" overhead in the old FORTRAN code includes the initial reduction of the system matrix to Hessenberg form which takes $O(n^3)$ FLOPs, while the "per frequency point" calculation takes $O(n^2)$ FLOPs.

### 3.1.3. Accuracy

No formal error analysis has been performed on the new algorithm. There is, however, numerical evidence to support the thesis that the new algorithm is more accurate than the older techniques, particularly when applied to larger systems.

Outputs from the four algorithm realizations were compared. For each frequency value, individual entries in the frequency response matrices computed by the four codes were compared using a symmetric relative error: The discrepancy between complex numbers $\eta$ and $\zeta$ (not both 0) was measured by

$$\delta(\eta,\zeta) = \frac{|\eta - \zeta|}{.5(|\eta| + |\zeta|)}.$$

e symmetric relative error uses the average magnitude of the two numbers being compared. It is intended for use in comparing two numbers neither of which was known to be correct.

This error measure ranges from a minimum of 0 to a maximum of 2. A value of $\delta(\eta,\zeta)$ near $10^{-n}$ indicates that $\eta$ and $\zeta$ agree to about $n$ decimal places while $\delta(\eta,\zeta) > .1$ indicates anything from rough approximation (near .1) to no correlation (bigger than, say, 1). For each fixed frequency, the worst discrepancy over all possible input-output pairs was observed.

The size of the discrepancy between the frequency response function matrices computed by these codes was observed to depend not only on which two of the codes were being compared but also on the size of the system, the frequency, and whether or not feedforward was present. It would take too much space to present details of these comparisons. However, some general statements can be made.

For each of the test problems (corresponding to one row of Table 1), the results produced by the four codes were compared two by two. The overall best agreement between any pair of calculations came from comparing the outputs of the new MEX-code and the new M-code. At worst, these agree to within 7 decimal places. This generally improves with reduction of system size or increase in frequency so that best agreement is within machine accuracy. No other pairing, either between one of the new codes and one of the old or between the two old, ever showed noticeably better agreement, and in general the agreement was much worse. Often, results from comparing the two new codes showed that the agreement of their computations was better than that of any other pairing by at least 2 decimal places. In the largest system, this advantage could increase to 5 decimal places, particularly for small frequencies or when no feedforward was present.

Thus, two dissimilar implementations of the new algorithm produce results in good agreement. When a parallel process is applied to the older algorithm, the two dissimilar implementations produce results which are not in such good agreement, either with each other or with those of the new algorithm.

To provide further evidence that the results of the new algorithm are more accurate, the old FORTRAN code was translated to quadruple precision from its native double precision and was run (at a time penalty of about $32\times$) on the medium sized problem using no feedforward and 200 frequency points. The output from this showed the same degree of agreement with the output from the new codes as they showed with each other.

These results combine to indicate strongly that the new algorithm provides more accurate results than those previously available. There are theoretical grounds for expecting this. The old way requires the solution of linear systems with the coefficient matrix $sI - \tilde{A}$ which is usually of large order. It also had conditioning problems which balancing ameliorated, but did not totally eliminate.

In the new algorithm, the coefficient matrices involved in the solution of linear systems are $\Delta$, which has the same order as the controller, and $sI - A_s^i$, which is of order 2, for $i = 1, \cdots, p$. Particularly when dealing with a large order structural model, the coefficient matrices used by the new algorithm are much smaller than the matrix $sI - \tilde{A}$ used by the old, so there is much less opportunity for round-off error. Any conditioning problems coming from the interaction of the frequency represented by $s = j\omega$ and the $i$–th structural mode in the old method is isolated in the new method to calculating the denominator term $\omega_i^2 - \omega^2 + 2j\zeta_i\omega_i\omega$ in equation (29); and this only gives numerical problems if $\omega$ is so close to $\omega_i$ that truncation occurs in forming the difference, and $\zeta_i$ is so small that the (small) real part $\omega_i^2 - \omega^2$ is a significant part of the whole term.

## 3.2. Test 2: Current Implementation

In Test 2, seven different closed-loop continuous time transfer functions are calculated using the current implementation of the software. It was deemed unnecessary to consider as many combinations of the parameters as was done in Test 1.

As noted in §2.11.2, the 16 closed-loop continuous transfer functions are calculated using one of five MATLAB M-files. Two of the five accept feedthrough matrices as an optional parameter and conditionally perform calculations which depend on the feedthrough matrices. The seven test cases are chosen so that all five M-files are exercised, and the two which accept feedthrough matrices are exercised both with and without the optional feedthrough matrices.

It was decided that the phenomenon of algorithm start-up time was adequately examined in Test 1. So, in Test 2, a 301 point frequency vector with the points distributed logarithmically between frequencies of .01 and 10000 radians per second was used for all cases.

Linear systems with two different numbers of state variables were used in calculating each of the seven transfer functions; one had the same number of states as the medium plant of Test 1 with 221 states (184 structural and 39 controller) and the other had the same number of states as the large plant of Test 1 with 1445 states (1406 structural and 39 controller). The numbers of inputs and outputs and the presence or absence of feedforward depended on the individual transfer function and are given in the following chart:

| Transfer function | Number of inputs | Number of outputs | Feedforward present |
|---|---|---|---|
| $T_{yr}$ | 6 | 6 | no |
| $T_{yw}$ | 10 | 6 | no |
| $T_{y^{pr}r}$ | 6 | 27 | yes |
| $T_{y^{pr}w}$ | 10 | 27 | yes |
| $T_{cr}$ | 6 | 6 | no |
| $T_{ur}$ | 6 | 3 | no |
| $T_{ud}$ | 3 | 3 | no |

### 3.2.1. Software Used in Test 2

In this study, two software realizations of the closed-loop continuous time frequency response function calculation algorithm given in this paper are compared to two software realizations of previously available frequency response function calculation algorithms and to one special purpose linear equation solver.

As noted in §2.11.2, the current software implementation of the calculation of the 16 closed-loop continuous time transfer functions is programmed as a collection of six MATLAB function M-files. These will be referred to as the new M-code. Minor modifications of these M-files are then translated into computer code in the C programming language using the MATLAB Compiler. The C code is then compiled into a MATLAB MEX-file using the MATLAB script `cmex`. This will be called the new MEX-code.

The same old FORTRAN code and old M-code used in Test 1 are used here for timing comparisons. Also, a special purpose linear equation solver due to W. Govaerts and J. D. Pryce [21] is used for both timing comparisons and accuracy assessment.

The Govaerts-Pryce (GP) algorithm assumes that the user has a matrix and further that the user knows how to solve linear systems using both that matrix and its transpose as the coefficient matrix. GP further assumes that what the user really wants is to solve linear systems using a larger matrix formed by adding some rows and columns to the given matrix. By using information from user provided solutions to linear systems with the smaller matrix and its transpose as coefficient matrices, GP calculates the solution to the larger system.

In Test 2, GP is used to calculate expressions of the form $\left(sI - \widetilde{A}\right)^{-1} \widetilde{B}$ (see §2.7). The block structure of the coefficient matrix $sI - \widetilde{A}$ shown in equation (36) is used. It is easy to solve equations using the upper left corner $E_{11}(s)$ or its transpose as a coefficient matrix since $E_{11}(s)$ is block diagonal with $2 \times 2$ blocks. Since the GP calculation is being included in the comparisons primarily as a baseline for accuracy checking, equations with these $2 \times 2$ blocks or their transposes as coefficient matrices are solved using full pivoting Gaussian elimination, which is at least as accurate as the Cramer's rule based solution process given in equation (27) for use in implementing the present algorithm.

### 3.2.2. Timing Comparisons

First, take note of comparisons which will *not* be made. Tests 1 and 2 were made about 2 years apart. While they were made on the same computer, there were subsequent hardware and software upgrades to the computer. This means that, even if the tests had been identical, the results would not have been comparable. Furthermore, while the same data set is used to generate test cases for both tests, none of the 12 cases used in Test 1 exactly matches any of the 14 cases used in Test 2. So, no comparison will be made between actual running times of any of the Test 1 cases with those of any of the Test 2 cases.

Evaluating seven different transfer functions using systems of two different sizes for each gives a total of 14 test cases. These cases were evaluated using the software described in the previous section with the exception that the old M-code was not applied to the large systems since this would have taken an unreasonable amount of computer time. The cases using M-code or MEX-code were timed using the MATLAB built in function cputime. The old FORTRAN code and the GP code were timed using Sun FORTRAN timing function DTIME. Results of the 63 individual timing runs are given in Table 2.

The current implementation of the algorithm continues to outperform traditional software, with the degree of improvement increasing with the size of the plant. On the medium sized test cases, the new M-code ran faster than the old M-code by a median factor of more than 34, while the median speed up of the new MEX-code over the old FORTRAN code was by a factor of more than 2.3. No attempt was made to run the old M-code on the large cases. On the seven large cases, the new MEX-code showed speed ups over the old FORTRAN code by factors ranging from 42 to 235 (median, 153).

The new MEX-code still outperforms the new M-code. In the medium sized cases, the median speed up of new MEX-code over new M-code is by a factor of 1.5 and in the large sized cases, the median speed up factor is 1.75. This speed up is not as good as in Test 1.

The difference in speed up factors is probably attributable to the way in which the respective MEX-codes were generated. In Test 1, the MEX-code was compiled from handwritten FORTRAN code. In Test 2, the MEX-code was compiled from C code which was generated from M-files using the comparatively recently released MATLAB utility mcc. So, the Test 1 code had a couple of advantages. It was hand written, which has at least the potential of producing more efficient code than an immature machine translation scheme. It was written in FORTRAN, and, in the authors' experience, contemporary FORTRAN compilers seem to do a better job of optimizing scientific calculation code than do contemporary C compilers. The decision to use the automatically generated code instead of hand-crafted code for the MEX-files in the current software implementation of the algorithm of this paper was an economic one based on available man-hours.

Although testing with the Govaerts-Pryce code was included primarily to assess the accuracy of the current implementation of the new algorithm, timing measurements were taken. The new MEX-code was also faster than the GP code, with median speed up factors of 8.9 for the medium sized plants and 40.6 for the large plants. So, while the GP code takes advantage of the sparsity in the structural plant matrix in its own way, the algorithm of this paper is still preferable for computational speed. While the old FORTRAN code was designed for efficiency in computing (full matrix) transfer functions at multiple frequency points, on the large plant, the GP code, with its exploitation of sparsity, was faster than the old FORTRAN code by a median speed up factor of 3.78.

### 3.2.3. Accuracy

Numerical evidence from the results of Test 2 continue to support the thesis that the new algorithm is more accurate, particularly on large problems, than traditional transfer function calculation techniques. When Test 2 was made, the availability of the GP software not only gave another independent calculation of transfer function matrices for use in comparison, but also gave a new tool for generating a measure of relative error.

As explained in Appendix B, the discrepancy between the results of calculating a transfer function

$$T = \tilde{C}\left(sI - \tilde{A}\right)^{-1}\tilde{B} + D$$

in two different ways is measured on a component by component basis relative to the following quantity which will be called a comparison magnitude:

$$\left|\tilde{C}\right|\left|\left(sI - \tilde{A}\right)^{-1}\right|\left|\tilde{B}\right| + |D|$$

The challenging part of computing this comparison magnitude, the determination of $\left(sI - \tilde{A}\right)^{-1}$, was accomplished using GP to solve the system of linear equations having $sI - \tilde{A}$ as a coefficient matrix and the identity matrix on the right hand side of the equation. All values for discrepancies and errors reported subsequently are relative to this comparison magnitude.

As in Test 1, unless otherwise mentioned, all computations were performed in 64 bit ANSI standard floating point computer arithmetic, called "double precision" on the Sun workstation used for this study. In Sun double precision arithmetic, machine epsilon, the smallest number which, when added to 1 in computer arithmetic, produces a result which is different from 1, is $2^{-52}$ which is about $2.2 \times 10^{-16}$.

In order to judge the accuracy of the GP code on the test problems, a quadruple precision (128 bit) version was created and exercised on the same test problems (using every 10*th* frequency point; the median over the 14 test cases of computation time per frequency point for quadruple precision was about 70 times that for double precision). Except for the two test cases calculating $T_{ur}$, the discrepancy between the double and quadruple precision results was never more than $10^{-14}$, and was usually less than $10^{-15}$. Even in the exceptional cases, the discrepancy never exceeded $10^{-12}$ and was usually less than $10^{-14}$. Because of these comparisons, the GP results were taken as the accuracy standard against which other results were measured.

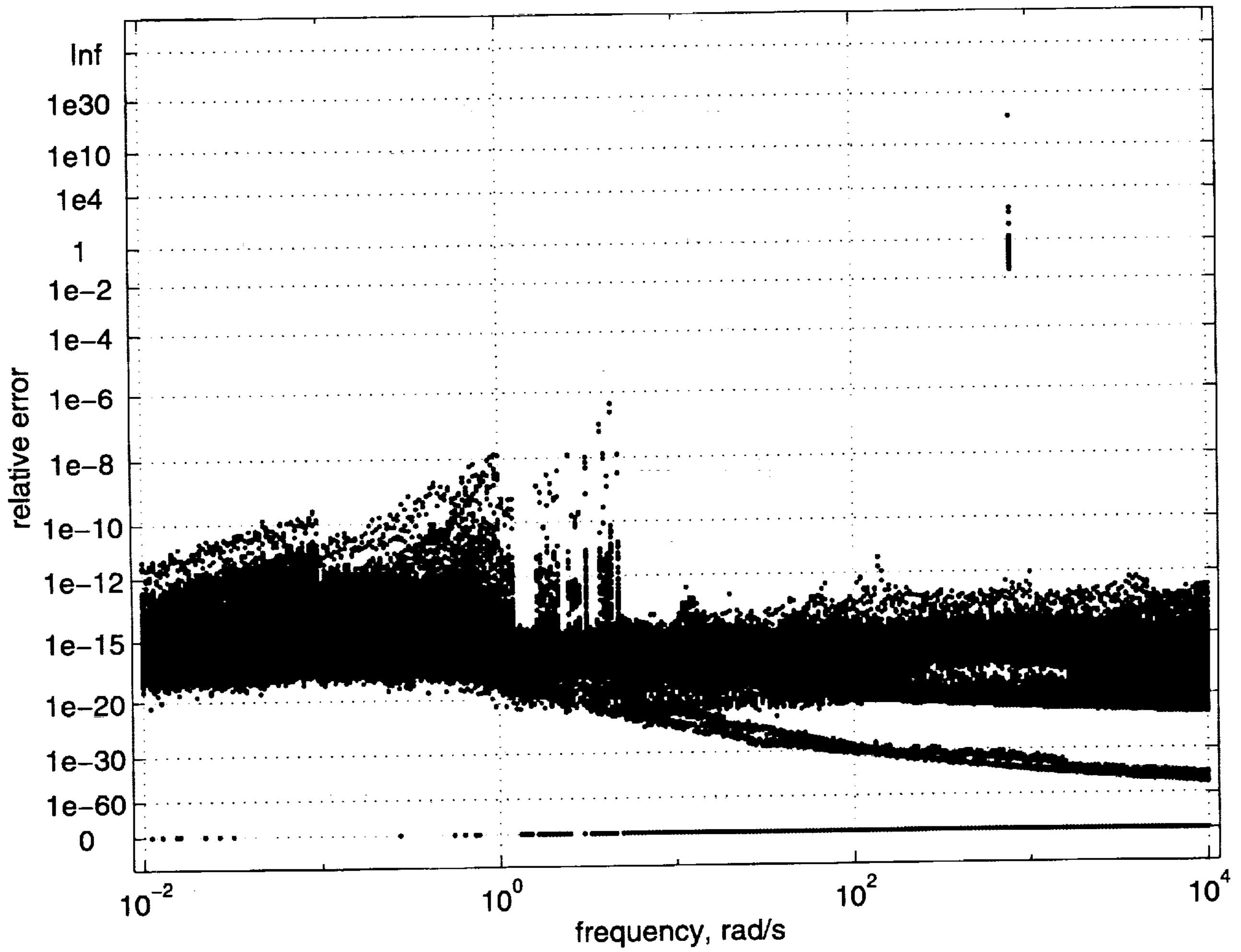

Figure 4. Discrepancy between the transfer function values calculated by the new MEX-code and the GP code, all test cases shown.

Of particular interest in this paper are results reflecting on the accuracy of the algorithms developed herein. The two software realizations of that algorithm, the new MEX-code and the new M-code, can be viewed as being compilations by two different compilers of essentially the same code. It is expected that they would give very similar results. Indeed, when compared, discrepancies between their results were uniformly bounded over the 14 test cases by $10^{-15}$ with a substantial proportion being identical. So, it is enough to judge the accuracy of the present algorithm to compare the results computed by the new MEX-code to the results computed by the GP code.

The 14 test cases each calculated transfer function matrices at 301 different values of frequency. These matrices had dimensions ranging from $3\times3$ to $10\times27$, depending on the case. A total of 355,782 data points were generated. A scatter plot showing the discrepancies between the values calculated by the new MEX-code and those calculated by the GP code at all 355,782 data points is given in Fig. 4. Of those, 88.69% of the new MEX-code answers showed an error (compared to the GP answers) of less than $10^{-14}$, 99.65% showed an error of less than $10^{-11}$, 99.984% showed an error of less than $10^{-8}$, and 99.987% showed an error of less than $10^{-6}$. The remaining 46 data points (.013%) showed errors of at least .01, with the worst case discrepancy being $2.8\times10^{17}$. These errors were troubling enough to warrant further investigation. Results of this investigation are reported at the end of this section.

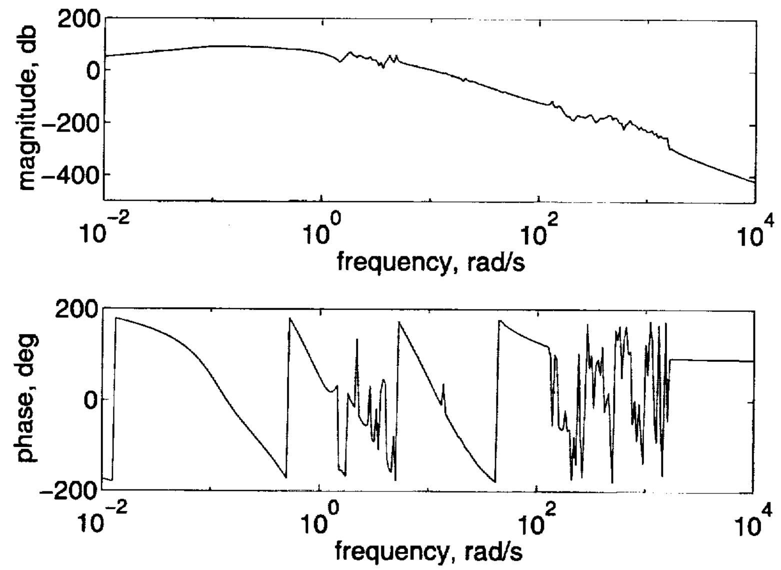

Figure 5. Transfer function matrix element $T_{pr}(13, 2)$ for the large plant calculated by the new MEX-code or the GP code.

The transfer function matrix element $T_{pr}(13, 2)$ for the plant with 703 structural modes is a typical result of these calculations. A Bode plot of this transfer function as calculated by the new MEX-code or the GP code is shown in Fig. 5. The same plot can be shown for both calculations since the discrepancy between them, as shown in Fig. 6, is below the plot resolution.

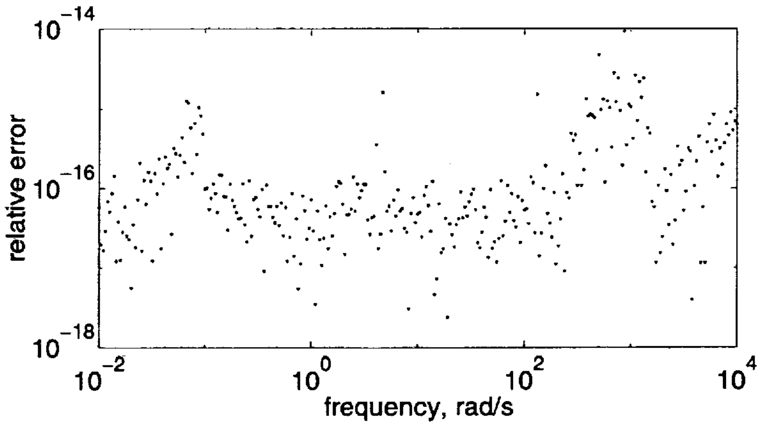

Figure 6. Discrepancy between the transfer function matrix element $T_{pr}(13, 2)$ for the large plant calculated by the GP code and the new MEX-code.

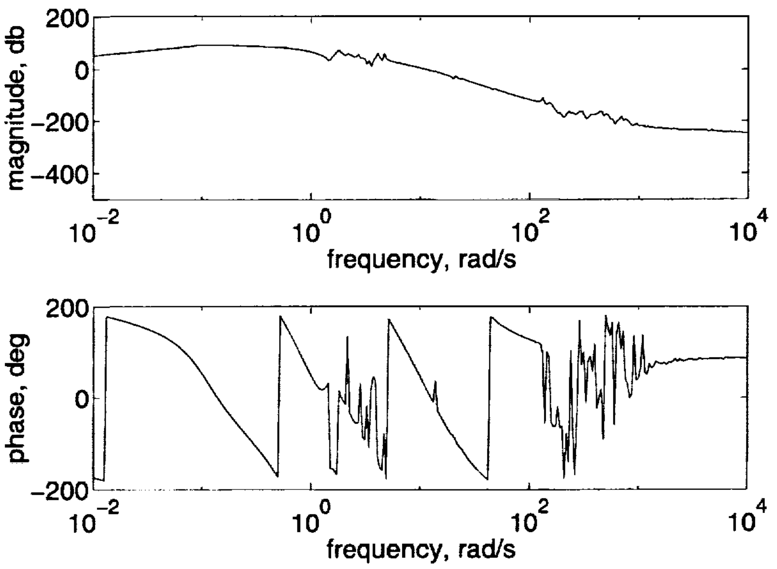

Figure 7. Transfer function matrix element $T_{pr}(13, 2)$ for the large plant calculated by the old FORTRAN code.

With respect to accuracy, the old FORTRAN code and the old M-code showed very similar accuracy behavior, so only the old FORTRAN code will be discussed. For two of the transfer functions, $T_{yw}$ and $T_{pw}$, accuracy of the old FORTRAN code was adequate, with errors of $10^{-7}$ or better. However, for the remainder of the transfer functions, the accuracy results were frequency dependent. Accuracy was good to adequate for frequencies between .01 and 1 rad/s. It then tended to degenerate as frequency increased from 1 to 10, then ramped up sharply so that when frequency reached $10^4$ rad/s, the size of errors ranged from $10^8$ to $10^{36}$.

Observe the same transfer function matrix element $T_{pr}(13, 2)$ for the same plant which was considered in connection with calculations using the GP and MEX-codes. A Bode plot of this transfer function as calculated by the old FORTRAN code is shown in Fig. 7. Over the last decade of frequency values, this plot diverges noticeably from the plot in Fig. 5. The magnitude of the discrepancy between these two calculations is shown in the scatter plot in Fig. 8.

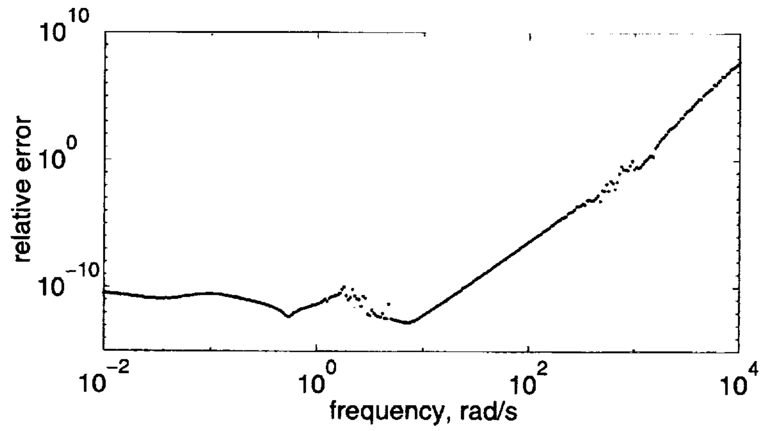

Figure 8. Discrepancy between the transfer function matrix element $T_{pr}(13, 2)$ for the large plant calculated by the GP code and the old FORTRAN code.

**The exceptionally large errors.**

Recall that a small fraction of the new MEX-code calculated data points, 46 out of 355,782 or .013%, showed errors of at least 0.01 when compared to the GP code calculations. The worst case discrepancy was $2.8 \times 10^{17}$. This point was truly exceptional. The next worst discrepancy was less than $1.6 \times 10^{2}$, and only a total of 7 errors were more than 1.

These larger errors occurred in several different transfer function evaluations ($T_{y^{pr}r}$, $T_{er}$, $T_{ur}$, and $T_{ud}$), but all in cases involving the 703 mode structural model, and all at the same frequency, 794.3282 rad/s. The worst error occurred in the (1,4) position of the transfer function matrix $T_{ur}$, so it was chosen for further analysis. Recall, Eq. (53), that

$$T_{ur}(s) = C_c \Delta^{-1}(s) B_c.$$

The baseline Bode plot for the transfer function matrix element $T_{ur}(1, 4)$ for the large plant as calculated by the GP code is shown in Fig. 9. When the same calculation is done by the new MEX-code, it results in the Bode plot shown in Fig.

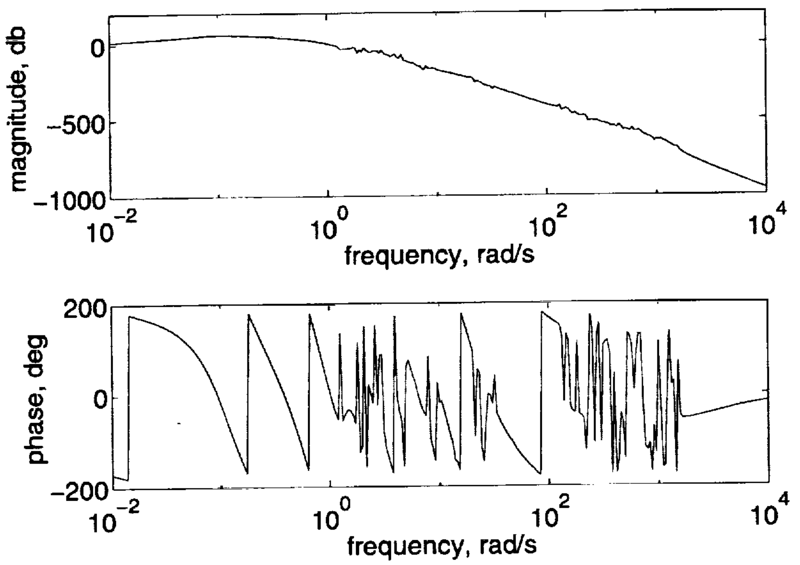

Figure 9. Transfer function matrix element $T_{ur}(1,4)$ for the large plant calculated by the GP code.

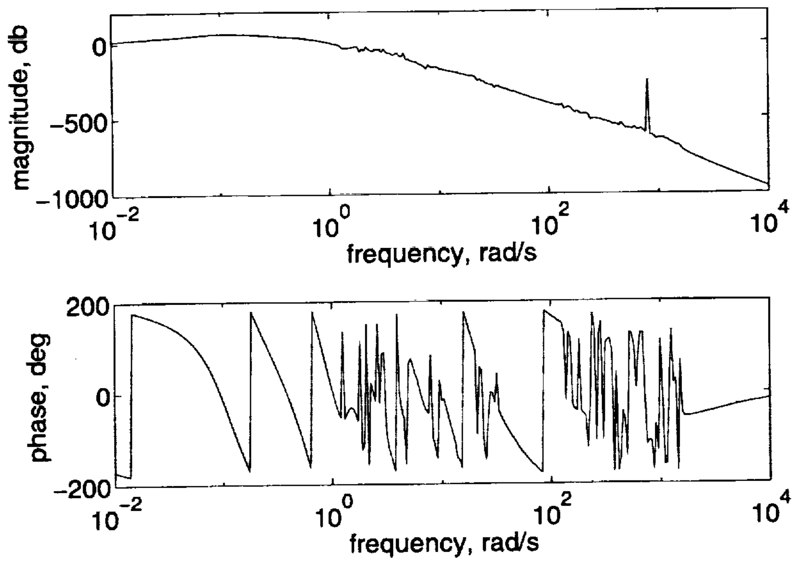

Figure 10. Transfer function matrix element $T_{ur}(1,4)$ for the large plant calculated by the new MEX-code.

10. The numerical problem shows up as a spike on the magnitude plot at about 800 rad/s. It can also be seen in the scatter plot of discrepancies between these two calculations shown in Fig. 11, where the exceptionally bad point is emphasized with an x.

Again, this is the exceptionally bad case. The spike on the next two worst

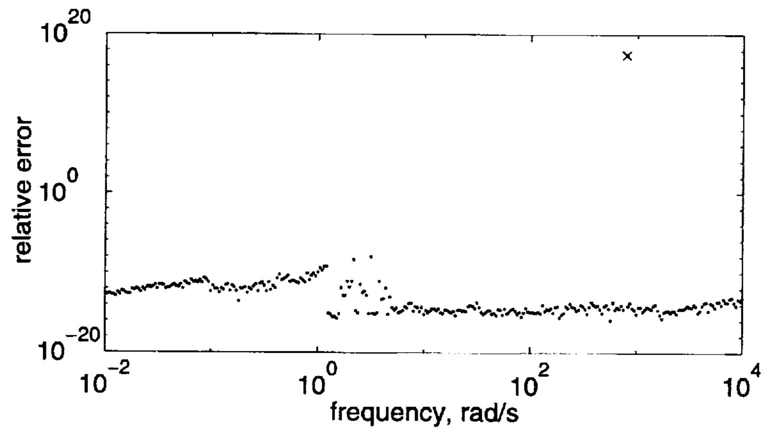

Figure 11. Discrepancy between the transfer function matrix element $T_{ur}(1,4)$ for the large plant calculated by the GP code and the new MEX-code.

cases would be about 1/8 as tall as this one, with the next worse after that being about 1/16 as tall.

The calculation of $T_{ur}(1,4)$ was repeated using the new MEX-code and the GP code and a grid of 30,001 points; i.e., 100 times the sampling density of the previous calculation. The error occurred at other points of this finer grid, but all close to the point already discovered. A detail plot of this area covering all additional erroneous points is shown in Fig. 12. The points from the magnitude plot of Fig. 10 are marked here with an x. The baseline GP calculation is plotted using a solid line, the new MEX-code calculation using a dotted line. Additional error points were found; the magnitudes of errors were all about the same and the frequencies at which they occur were limited to the range from about 750 rad/s to 810 rad/s.

The actual calculation of $T_{ur}(s)$ was being done by a MATLAB statement of the form:

```
>> T = Cc*(DELTA\Bc);
```

Here, the MATLAB variables `Cc` and `Bc` hold the values of the matrices $C_c$ and $B_c$, respectively, while the MATLAB variable `DELTA` holds the value of the matrix $\Delta(s)$ with $s = 794.3282j$. The (1,4) element of $T_{ur}$ can be considered as being the result of the MATLAB calculation

```
>> T(1,4) = c*(DELTA\b);
```

where `c` is row 1 of `Cc` and `b` is column 4 of `Bc`. A wide range of magnitudes was observed in this calculation. The calculated values in the column vector `DELTA\b` varied over about 47 orders of magnitude. When the product `c*(DELTA\b)` was

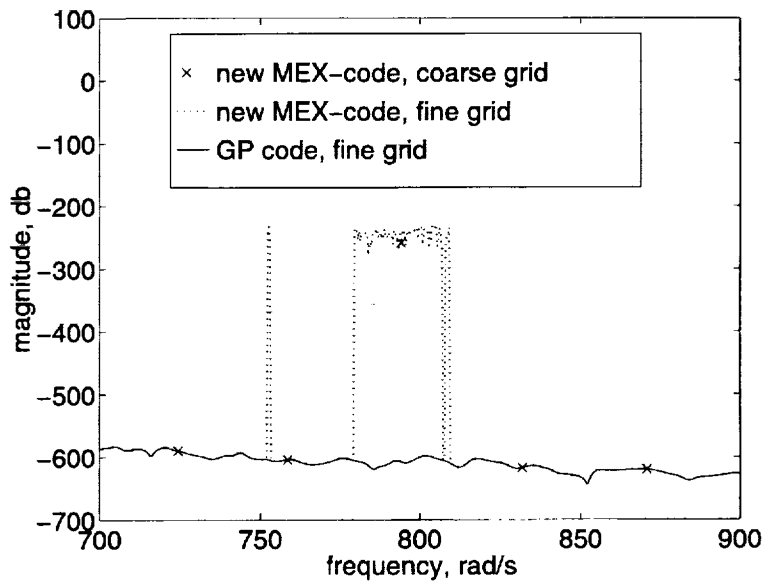

Figure 12. Detail of fine grid calculation of transfer function matrix element $T_{ur}(1,4)$ for the large plant.

formed, zeros in the row vector `c` canceled out elements of the top 16 orders of magnitude in `DELTA\b`. Thus, for `T(4,1)` to be accurate, small components of `DELTA\b` needed to be computed accurately, and in the case under study, this did not happen.

Several experiments were conducted to see if improving the linear equation solution process implicit in the MATLAB computation `DELTA\Bc` would give values of $T_{ur}$ which agree with the GP solution. Two were successful at all but one data point; that one data point being the same (1,4) element of $T_{ur}$ at frequency 794.3282 rad/s considered previously. When two cycles of iterative improvement were used to improve the accuracy of the computed value for $\Delta^{-1}B_c$, except for the one point noted, the results came into agreement with the results produced by the GP code. The same agreement was achieved when $\Delta^{-1}B_c$ was computed using the linear equation solution subroutine `ZGESVX` from LAPACK [15]. Even for the one bad point, the error decreased from $2.8\times10^{17}$ to $2.0\times10^{2}$. So, the problem of the few inaccurate points was traced to numerical behavior of the MATLAB linear equation solver and could generally be relieved by using a more robust linear equation solver to calculate expressions involving $\Delta^{-1}$.

When the complete suite of tests was recalculated with the new MEX-code modified to use LAPACK subroutine `ZGESVX` for all computations involving $\Delta^{-1}$, the discrepancies of the results, when compared to those of the GP code, are shown

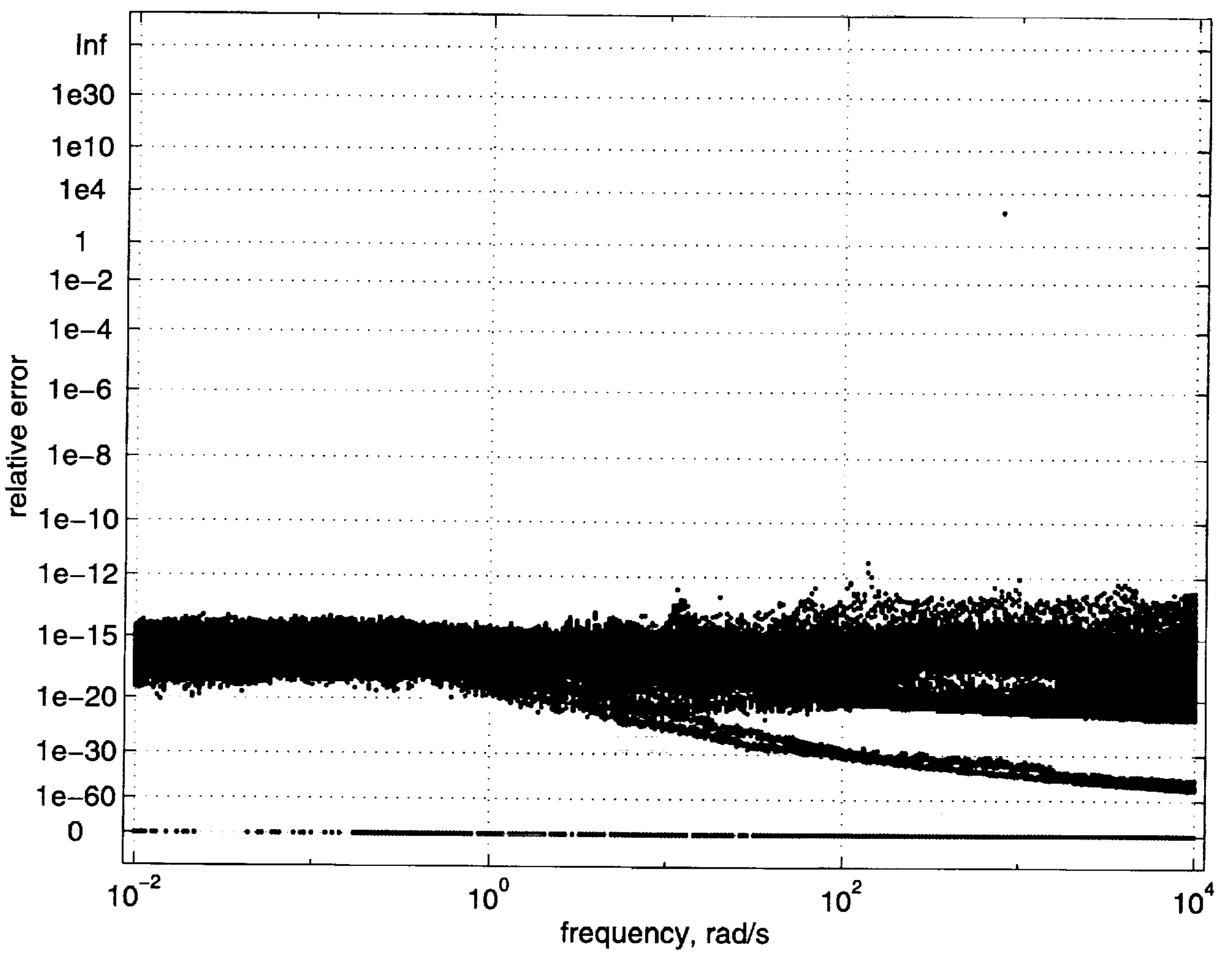

Figure 13. Discrepancy between the transfer function values calculated by the new MEX-code with ZGESVX and the GP code, all test cases shown.

in Fig. 13. By comparing this figure to Fig. 4, it can be seen that not only has the worst point improved and the rest of the violators of the $10^{-2}$ relative error magnitude line been reduced to levels consistent with the GP results, but also there has been a general improvement in accuracy in the lower frequencies. The down side to the accuracy improvement using ZGESVX is that the median time over the 14 cases to calculate the transfer function matrices using ZGESVX is 1.7 times that of using the MATLAB matrix division operator.

It was also observed that this phenomenon occurred at a frequency where the affected transfer function values had experienced a large amount of roll-off. From a point of view of engineering analysis, even if these inaccuracies remained in the calculation, they would probably have no impact on any engineering decisions made on the basis of this calculation.

# 4. Concluding Remarks

An efficient and novel procedure has been developed for the calculation of the frequency response function of a large order, flexible system implemented with a linear, time invariant control system. The procedure takes advantage of the highly structured sparsity of the system matrices of the plant in normal mode coordinates. This reduces the computational cost from a quadratic function of the order of the system to a linear function, thereby permitting the practical frequency analysis of systems of much larger order than by traditional, full-matrix means. Formulations have been given for both open and closed-loop systems for both continuous time and discrete time formulations.

Numerical examples were presented wherein the advantages of the present algorithm over traditional approaches, both in speed and in accuracy, have been demonstrated. When the initial software implementation of the algorithm was exercised on the largest systems, the MATLAB MEX-file based on a FORTRAN implementation of the new algorithm (the new MEX-code) was about 40 times faster than the old FORTRAN code when many frequency points were used while the advantage increased to a factor of about 80 or better when the calculation involved few frequency points. In this latter case, MATLAB M-files based on the new algorithm (the new M-code) was over 200 times faster than the old M-code. On the largest systems, the current MEX-code implementation of the algorithm, C language code generated automatically from MATLAB M-files by the MathWorks `mcc`, was faster than the old FORTRAN code by factors ranging from 42 to 235 depending on which input/output pair and transfer function matrix size were chosen. Even though the Govaerts-Pryce (GP) code, which was used for an accuracy baseline, took advantage of the same sparsity in the plant system matrix as did the new MEX-code, the new MEX-code was faster than the GP code by a median factor of 8.9 for medium sized plants and by a median factor of 40.6 for large sized plants.

In this study, the standards of accuracy were set by the GP code. With only a tiny number of exceptions (0.013% of all data points), results calculated by the new MEX-code and the new M-code agreed with the GP code results to at least 8 decimal places. The exceptionally large errors were traced to problems experienced by the MATLAB linear equation solver with certain of the $\Delta$ matrices for coefficient matrices. By contrast, at a substantial fraction of data points, the results of the old FORTRAN code and the old M-code disagreed with those of the

GP code by totally unacceptable error amounts, with discrepancies ranging up to many orders of magnitude.

# Appendix A: Faster Calculation of $C_c\Delta^{-1}(s)B_c$

The expression $C_c\Delta^{-1}(s)B_c$ occurs (directly or indirectly) in the formulae for all 16 of the closed loop transfer functions presented in this paper. If standard full matrix techniques are used to calculate this expression (the product $\Delta^{-1}(s)B_c$ is calculated by solving a linear system with $\Delta(s)$ as the coefficient matrix and $B_c$ as the right hand side and the calculation is completed by a matrix multiplication), then the FLOP count for this calculation is $O(k^3 + k^2f + kmf)$. This appendix presents a technique which can be used to replace the $k^3$ term by $k^2m$ if transfer functions must be calculated at enough (order of $k/(f+m)$ or more) different values of $s$.

First, some mathematical machinery must be developed; then it is applied to this specific problem.

## A.1. Solving Low Rank Modifications of Linear Systems

In this section, it is shown how to express the solution of one system of linear equations in terms of the solution of another system of linear equations whose coefficient matrix is a low rank perturbation of the coefficient matrix of the original system. The conventions which have been established in this paper for the use of symbols are suspended for the remainder of this section in favor of:

**Notation:**

| | |
|---|---|
| $I$ | Identity matrix of size appropriate to context |
| All other capital letters | Matrices over some field |
| $x$ and $y$ | vectors over some field |
| All other lower case letters | Positive integers |

The results of this section are valid for matrices with entries from an arbitrary field. For engineering purposes, the most important fields are the real and complex number fields.

In this section, $A$ represents an arbitrary non-singular matrix. The results of this section are most useful when $A$ has the property that linear systems using $A$ for a coefficient matrix can be solved more easily than would be the case for an arbitrary matrix of the same size as $A$. For example, $A$ could have a known LU decomposition (factorization into lower triangular and upper triangular matrices), $A$ could be block diagonal, or (the case of interest in this paper), $A$ could be upper Hessenberg (i.e., $A$ is zero below the first subdiagonal).

The main result of this section depends on a lemma.

**Lemma:** Let $A$ be a non-singular $k \times k$ matrix, and let $R$ and $S$ be $k \times r$ matrices. Then, $A + RS^T$ is non-singular if and only if $I + S^T A^{-1} R$ is non-singular.

Proof: Both parts of the "if and only if" will be proven by contradiction.

Suppose that $A + RS^T$ is singular. Then, there exists an $x \neq 0$ for which $(A + RS^T)x = 0$. Then, $Ax = R(-S^T x)$. Since $A$ is non-singular, $Ax \neq 0$, and so $-S^T x \neq 0$. If $I + S^T A^{-1} R$ is applied to this nonzero vector, the result is 0:

$$\begin{aligned}\left(I + S^T A^{-1} R\right)\left(-S^T x\right) &= -S^T x + S^T A^{-1} A x \\ &= -S^T x + S^T x = 0\end{aligned}$$

This demonstrates that $I + S^T A^{-1} R$ is singular. This establishes the "if" portion of the Lemma.

Now, suppose that $I + S^T A^{-1} R$ is singular. Then there exists a $y \neq 0$ for which $(I + S^T A^{-1} R)y = 0$. Then, $y = S^T(-A^{-1} R y)$. Therefore, $-A^{-1} R y \neq 0$. If $A + RS^T$ is applied to this nonzero vector, the result is 0:

$$\begin{aligned}\left(A + RS^T\right)\left(-A^{-1} R y\right) &= -Ry + RS^T\left(-A^{-1} R y\right) \\ &= -Ry + Ry = 0\end{aligned}$$

This demonstrates that $A + RS^T$ is singular. This establishes the "only if" portion of the Lemma. $\square$

The *Low Rank Modification Theorem for Solution of Linear Equations*, which is also a corollary to the *Sherman-Morrison-Woodbury formula* [6, p. 51] or [22, 124], can now be stated:

**Theorem:** Let $A$ be a $k \times k$ matrix, $R$ and $S$ be $k \times r$ matrices, and $B$ be a $k \times m$ matrix. Suppose that $A$ and $A + RS^T$ are non-singular. Then there exist a $k \times r$ matrix $Z$ and $k \times m$ matrices $Y$ and $W$ satisfying, respectively, the equations $AZ = R$, $AY = B$, and $(I + S^T Z)W = S^T Y$. Further, if $X = Y - ZW$, then $X$ satisfies the equation $(A + RS^T)X = B$.

Remark: Note that $A + RS^T$ is perturbed from $A$ by a matrix of rank at most $r$. This theorem says that one can solve $m$ linear equations with the order $k$ coefficient matrix $A + RS^T$ by solving $m + r$ linear equations with the order $k$ coefficient matrix $A$ and $m$ linear equations with the order $r$ coefficient matrix $I + S^T Z$. If the matrix $A$ has properties which allows one to solve equations with coefficient matrix $A$ more rapidly than those with coefficient matrix $A + RS^T$, and if $r$ is substantially smaller than $k$, and if $m$ is not too large, then it might be significantly more efficient to calculate $X$ as $X = Y - ZW$ than to calculate $X$ by solving $(A + RS^T)X = B$ directly.

Proof: The non-singularity of $A$ insures the existence of $Z$ and $Y$ with the stated properties. Since $I + S^T Z = I + S^T A^{-1} T$, the lemma insures that $I + S^T Z$ is non-singular, so there exists a $W$ with the stated property. Let $X = Y - ZW$. It follows from the equation defining $W$ that $S^T ZW = S^T Y - W$. Therefore,

$$\begin{aligned}\left(A + RS^T\right)X &= \left(A + RS^T\right)(Y - ZW)\\ &= AY - AZW + RS^T Y - RS^T ZW\\ &= B - RW + RS^T Y - R\left(S^T Y - W\right)\\ &= B\end{aligned}$$

□

## A.2. Application to Calculation of $C_c\Delta^{-1}(s)B_c$

The idea presented by Alan Laub in [1, 2], as applied to calculating the transfer function of the control system, would replace the control system equations (8) by

$$\begin{aligned}\dot{x}_h &= A_h x_h + B_h e\\ u &= C_h x_h\\ e &= r - y - v\end{aligned}$$

where $A_h = T^{-1} A_c T$, $B_h = T^{-1} B_c$, and $C_h = C_c T$; and $T$ is an orthonormal matrix chosen so that the matrix $A_h$ is upper Hessenberg (has zeros below the first subdiagonal). Essentially, this just performs a change of basis in the controller state space without changing the controller itself. The computational burden of performing this transformation is $O(k^3)$ FLOPs, but it needs to be done only once even if transfer functions are to be evaluated for multiple values of $s$.

If this transformation is not done, then to calculate the transfer function of the controller, it is necessary to evaluate $(sI - A_c)^{-1} B_c$ for each $s$. Computing this expression requires $O(k^3 + k^2 f)$ FLOPs. If, however, the transformation is performed, then, along with $A_h$, $sI - A_h$ is also upper Hessenberg, and the corresponding evaluation of $(sI - A_h)^{-1}B_c$ requires only $O(k^2 f)$ FLOPs.

It was already suggested in the discussion of computing open-loop transfer functions that this technique be applied to computing the open-loop expression

$$C_c X_{22}(s) B_c = C_c E_{22}^{-1}(s) B_c = C_c(sI - A_c)^{-1} B_c = C_h(sI - A_h)^{-1} B_h$$

Operationally, the simplest way to accomplish that is to replace the original representation of the control system by its equivalent representation using an upper Hessenberg "A" matrix.

The problem with calculating closed-loop transfer functions is that the open-loop relation $X_{22} = E_{22}^{-1}$ is replaced, in the closed-loop case, by the relation $X_{22} = \Delta^{-1}$. Now, recall from Eq. (40) that

$$\Delta(s) = E_{22}(s) + B_c C_s E_{11}^{-1}(s) B_s C_c$$

Furthermore the rank of the term added to $E_{22}$ to get $\Delta$ is at most $\min(f, m)$. It is now a small step to write $\Delta = E_{22} + RS^T$ where either $R = B_c$ and $S^T = C_s E_{11}^{-1}(s) B_s C_c$ or $R = B_c C_s E_{11}^{-1}(s) B_s$ and $S = C_c$ depending on whether $f$ or $m$ were smaller, respectively; and apply the Low Rank Modification Theorem for Solution of Linear Equations to the calculation of $\Delta^{-1}(s)B_c$. If $A_c$ is upper Hessenberg, so that the solution of linear equations with coefficient matrix $E_{22}(s)$ can be calculated substantially faster than for arbitrary $k \times k$ matrices, then the calculation of $\Delta^{-1}(s)B_c$ will be accomplished with fewer FLOPs than if full matrix techniques were used. The calculation of $C_c\Delta^{-1}(s)B_c$ is now completed with a matrix multiplication.

## A.3. Comparative Computational Complexity

In this section, the computational burden of computing $C_c\Delta^{-1}(s)B_c$ by standard full matrix techniques is compared to the computational burden of computing the same expression using the low rank modified Hessenberg technique of the previous section. Specifically, FLOP count expressions will be developed for the amount of calculation required to pass from the point at which $C_s E_{11}^{-1}(s) B_s$ has been calculated to the point at which the calculation of $C_c\Delta^{-1}(s)B_c$ is completed.

**Full matrix FLOP count**

In this FLOP count, it is assumed that $A_c$ and, therefore, $E_{22}(s) = sI - A_c$ are full matrices.

The term $B_c C_s E_{11}^{-1}(s) B_s C_c$ can be calculated from $C_s E_{11}^{-1}(s) B_s$ in $O(kfm + k^2 m)$ FLOPs by doing the $B_c$ multiplication first. The addition of $E_{22}$ to form $\Delta$ does not add significantly to the FLOP count. Calculating a partially pivoted LU decomposition of $\Delta$ requires $O(k^3)$ FLOPs. Using this LU decomposition, the calculation of $\Delta^{-1}(s) B_c$ can be completed in $O(k^2 f)$ FLOPs. The matrix multiplication required to calculate $C_c \Delta^{-1}(s) B_c$ requires $O(kfm)$ FLOPs. Thus, the total FLOP count to calculate $C_c \Delta^{-1}(s) B_c$ from $C_s E_{11}^{-1}(s) B_s$ using full matrix methods is $O(k^3 + k^2(f + m) + kfm)$. In the expected applications of this algorithm, $k$ is significantly larger than $f$ or $m$, so the dominant term of this FLOP count formula is $O(k^3)$.

**Low rank modified Hessenberg FLOP count**

In this FLOP count, it is assumed that $A_c$ and, therefore, $E_{22}(s) = sI - A_c$ are upper Hessenberg matrices. The FLOP count will be based on applying the Low Rank Modification Theorem for Solution of Linear Equations (LRMTSLE). These expressions will be developed under the assumption that $m \leq f$. If $f < m$, the alternate expressions given earlier for $R$ and $S$ are used, and the roles of $m$ and $f$ are reversed in the FLOP count expression, but this does not alter the dominant terms.

In order to express $\Delta$ in the form $E_{22} + RS^T$ needed to apply LRMTSLE, $R$ is taken to be the $k \times m$ matrix $B_c C_s E_{11}^{-1} B_s$ at a computational cost of $O(kfm)$, and $S^T$ is taken to be $C_c$ at no computational cost. Because $E_{22}$ is upper Hessenberg, the (now $k \times m$) matrix $Z$ of LRMTSLE can be calculated as the solution to the equation $E_{22} Z = R$ in $O(k^2 m)$ FLOPs. The (now $k \times f$) matrix $Y$ of LRMTSLE can be calculated as the solution to the equation $E_{22} Y = B_c$ in $O(k^2 f)$ FLOPs. The matrices $S^T Z$ and $S^T Y$ needed to calculate $W$ can be calculated in $O(km^2 + kmf)$ FLOPs. The calculation of the $m \times f$ matrix $W$ as the solution of the equation $(I + S^T Z) W = S^T Y$ must be done by full matrix techniques. This requires $O(m^3 + fm^2)$ FLOPs. The matrix $X$ of LRMTSLE, which *is* $\Delta^{-1}(s) B_c$, is calculated as $X = Y - ZW$ in $O(kfm)$ FLOPs. The matrix multiplication required to calculate $C_c \Delta^{-1}(s) B_c$ requires $O(kfm)$ FLOPs. Thus, the total FLOP count to calculate $C_c \Delta^{-1}(s) B_c$ from $C_s E_{11}^{-1}(s) B_s$ using the low rank modified Hessenberg method is $O(k^2(m+f)+k(fm+m^2)+fm^2+m^3)$. In the expected applications of this technology, $k$ is significantly larger than $f$ or $m$, so the dominant term of this FLOP count formula is $O(k^2(m+f))$. With the expected values of the parameters, this is a significant improvement over the full matrix formula.

This still leaves the question of how the total computation time is affected by adding in the time necessary to reduce an arbitrary controller representation to one in which the system matrix is in upper Hessenberg form. The general answer is that if there are enough distinct frequency points over which to amortize the cost of the Hessenberg reduction, then adding the cost of this initial reduction to the cost of computing the transfer function for all frequency values still results in a cheaper calculation than using the full matrix method. But, that brings up the question of how many frequency points are enough. One answer to this is given by observing that if there are $k/(f+m)$ or more points, amortizing the dominant $O(k^3)$ term of the start-up cost over these points contributes an additional $O(k^2(f+m))$ or less to each. This absorbs into the existing dominant term. Another answer can be given by using explicit FLOP counts given in [1]. The dominant term in the FLOP count for the orthogonal Hessenberg reduction (the more numerically stable of two techniques mentioned in [1]) is $5k^3/3$. The dominant term in the FLOP count for the full matrix solution method is $k^3/3$. This suggests that the break-even point for using the low rank modified Hessenberg method is that the calculation needs to be done for at least five different values of $s$ (i.e. at least five different frequencies). In typical applications (such as making a Bode plot), there are many more than five frequencies at which the transfer function is evaluated, so using the low rank modified Hessenberg method should give the answers with fewer FLOPs.

# Appendix B: Numerical Error Measurement: The Data Relative Error

Numerical calculations are subject to many kinds of error. Typically, only a fixed precision, a fixed number of decimal places or binary bits, are carried through the calculation, so round-off becomes a source of error. If two nearly equal numbers of a given precision are subtracted, the cancellation of high order digits or bits results in an answer of lower precision. An algorithm which would, theoretically, require infinitely many steps to arrive at the correct answer is terminated after only a finite number of steps producing a truncation error. In a long calculation, such errors can be cumulative, and can be magnified by ill conditioning of the problem.

In order to experimentally estimate the magnitude of such error in some given calculation, one might find oneself comparing two numbers to see if they are close together. For example, if one has performed a computation which is supposed to have solved an equation, one might substitute the computed candidate for a solution back into the equation, and see how close the left and right sides are. Or, one might use two (or more) different algorithms to approximate the same result and compare these approximations with each other. The question then is whether two numbers are "close" to each other.

There is no absolute answer to this question. For example, the absolute error between 15,995 and 15,999, i.e., the magnitude of their difference, is more than the absolute error between .0011 and .0023. But most people would judge that the numbers of the first pair are almost the same while the numbers of the second pair are substantially different. This is based on looking at the relative error, the absolute difference of the numbers relative to some appropriate magnitude such as their sizes. The symmetric relative error measure, $\delta$, introduced in §3.1.3, reflects this judgement, since $\delta(15995, 15999) \approx 0.00025$ while $\delta(.0011, .0023) \approx 0.71$.

The problem in calculating a relative error between two numbers becomes one of finding the comparison magnitude, a quantity of an appropriate magnitude to which the absolute error may be compared. If one of the numbers is known to be the correct one, and it is not zero, then its magnitude may be used; this is what is commonly known as *relative error*. The symmetric relative error uses the average magnitude of the two numbers being compared. It is intended for use in comparing two numbers neither of which was known to be correct.

However, $\delta$ is not always a good measure of whether two numbers which are supposed to be approximately the same really are. For example, suppose $x_a$ is an approximate (e.g., calculated) solution to the simple scalar linear equation $ax = b$ which can also be written $ax - b = 0$. If $x_a$ is substituted into either of the equations for which it is supposed to be a solution, one expects the left and right sides to be close to each other. Suppose that the result of the machine calculation $ax_a$ is not exactly $b$, and suppose that $\delta$ is used to measure this closeness. Using the first equation, the closeness will be measured as $\delta(ax_a, b)$, which will probably be on the order of machine epsilon. However, using the second equation, the closeness will be measured as $\delta(ax_a - b, 0) = 2$, which is the maximum measure of discrepancy which the symmetric relative error measure can return. This second test has given a false sense of the quality of $x_a$ as a solution to the equation $ax - b = 0$.

A problem with trying to measure how good an approximation $x_a$ is as a solution to the equation $ax - b = 0$ using $\delta(ax_a - b, 0)$ is that this symmetric relative error calculation does not take into account the magnitude of the data which went into calculating $x_a$. One way to take data size into account would be to measure the error relative to the size of the constant term $b$. As long as $b \neq 0$, the relative error measure becomes $|ax_a - b|/|b|$; the error is being measured relative to the magnitude of the constant term of the equation.

This actually gives a good indication of how close the residual $ax_a - b$ is to 0. However, if the scalar equation $ax = b$ is replaced by a system of equations represented in matrix-vector form as $Ax = b$, the natural extension of this relative error measure may not give such a good idea of whether all components are near 0. This extension would be to take $||Ax_a - b||/||b||$ as a measure of the error, where $x_a$ is again an approximate (e.g., computed) solution to the equation and $||\cdot||$ represents some vector norm. This error measure can be viewed as looking at the component by component size of the residual $Ax_a - b$ relative to $||b||$. This measure gives an accurate idea of the size of those components of the residual corresponding to the large components of $b$ (i.e., those components of $b$ which contribute significantly to $||b||$). However, some small components of $b$ may be poorly approximated by the corresponding components of $Ax_a$ without this error being reflected in the error measure $||Ax_a - b||/||b||$. Using $||b||$ as a comparison magnitude has masked the error in these small components. If one attempts to regain the lost information by looking at the size of each component of the residual vector relative to the magnitude of the corresponding component of $b$, one runs the risk of overestimating the error represented by components of the residual corresponding to small components of $b$ (to see the extreme case, imagine that some components of $b$ are zero).

This motivates the introduction of the notion of the *data relative error*. This is an error measure which is made on a component by component basis relative to comparison magnitudes which are based on all the data which went into the calculation of each component. In its ideal form, it is applicable to any arithmetic calculation producing a number which is theoretically supposed to be zero. For example, once an approximate solution $x_a$ to a linear system of equations $Ax = b$ is known, the calculation $Ax_a - b$ of the residual has these properties. A calculation is made parallel to the calculation which produced the "should be zero" quantity. In the parallel calculation, each datum originally used is replaced by its absolute value, and all subtractions are replaced by additions. For the linear equation example, this parallel calculation is $|A||x_a| + |b|$, where all absolute values are taken component by component.

Every individual scalar result $x$ of the original calculation has its counterpart $\xi \geq 0$ in the parallel computation, and (at least in exact arithmetic) $|x| \leq \xi$. The magnitude of $\xi$ reflects the magnitude of the data which went into the calculation of $x$ and gives an upper limit on what magnitude the answer could possibly have based on the form of the calculation. The calculation of $\xi$ does not take into account any cancellation, such as by the subtraction of nearly equal quantities, which might have occurred in the calculation of $x$. So, if $\xi > 0$, the ratio $|x|/\xi$ gives a reasonable idea of how close $x$ is to 0 by comparison with the data from which it was calculated and what the calculation process could potentially do to that data.

Unfortunately, the amount of programming necessary to calculate this ideal data relative error can be too large to be practical. For example, if $T^{(a)}\left(s, \tilde{A}, \tilde{B}, \tilde{C}, \tilde{D}\right)$ and $T^{(b)}\left(s, \tilde{A}, \tilde{B}, \tilde{C}, \tilde{D}\right)$ represent the results of computing the transfer function $\tilde{C}\left(sI - \tilde{A}\right)^{-1}\tilde{B} + \tilde{D}$ in two different ways, then the quantity which is supposed to be close to zero is

$$T^{(a)}\left(s, \tilde{A}, \tilde{B}, \tilde{C}, \tilde{D}\right) - T^{(b)}\left(s, \tilde{A}, \tilde{B}, \tilde{C}, \tilde{D}\right).$$

Computing the ideal data relative error would involve rewriting all of the software involved in calculations (a) and (b) to include parallel upper bound computations.

For purposes of Test 2 in this paper, a compromise was adopted. The comparison magnitude was taken from the computation

$$\left|\tilde{C}\right|\left|\left(sI - \tilde{A}\right)^{-1}\right|\left|\tilde{B}\right| + \left|\tilde{D}\right|.$$

The programming effort required to adapt the GP software to calculate $\left(sI - \tilde{A}\right)^{-1}$ was moderate; the computational time to calculate the inverse was reasonable; and the results were reliable, even when the order of the matrix was 1445. The remainder of the calculation was routine.

This comparison magnitude no longer has some of the properties of the ideal. For example, even if it were doubled to provide contributions for both transfer function calculations, the inequality $|x| \leq \xi$ between an individual calculated number and its comparison magnitude no longer necessarily holds. This is because $\left|\left(sI - \tilde{A}\right)^{-1}\right|$ is not an ideal comparison magnitude for $\left(sI - \tilde{A}\right)^{-1}$. And the compromise comparison magnitude does not necessarily refer to the actual algorithms used in computing either $T^{(a)}$ or $T^{(b)}$. Despite this, the compromise comparison magnitude performed very well in assessing questions of accuracy in the present study.

## References

[1] Laub, Alan J.: Efficient Multivariable Frequency Response Computations. *IEEE Transactions on Automatic Control*, vol. AC-26, no. 2, 1981, pp. 407–408.

[2] Laub, Alan J.: ALGORITHM 640. Efficient Calculation of Frequency Response Matrices from State Space Models. *ACM Transactions on Mathematical Software*, vol. 12, no. 1, 1986, pp. 26–33.

[3] Maghami, Peiman G.; and Giesy, Daniel P.: Efficient Computation of Closed-Loop Frequency Response for Large-Order Flexible Systems. *Journal of Guidance, Control, and Dynamics*, vol. 19, no. 4, 1996, pp. 780–786.

[4] Maghami, Peiman G.; Kenny, Sean P.; and Giesy, Daniel P.: *PLATSIM: An Efficient Linear Simulation and Analysis Package for Large-Order Flexible Systems*. NASA TP 3519, 1995.

[5] Ogata, Katsuhiko: *Modern Control Engineering*. Prentice Hall, Second ed., 1990.

[6] Golub, Gene H.; and Van Loan, Charles F.: *Matrix Computations*. The Johns Hopkins University Press, Second ed., 1989.

[7] *MATLAB High-Performance Numeric Computation and Visualization Software. Reference Guide*. The MathWorks, Inc., Natick, MA, August 1992.

[8] Lawson, C. L.; Hanson, R. J.; Kincaid, D. R.; and Krogh, F. T.: Basic Linear Algebra Subprograms for Fortran Usage. *ACM Trans. Math. Soft.*, vol. 5, no. 3, 1979, pp. 308–323.

[9] Lawson, C. L.; Hanson, R. J.; Kincaid, D. R.; and Krogh, F. T.: Algorithm 539. Basic Linear Algebra Subprograms for Fortran Usage [F1]. *ACM Trans. Math. Soft.*, vol. 5, no. 3, 1979, pp. 324–325.

[10] Dongara, J. J.; Moler, C. B.; Bunch, J. R.; and Stewart, G. W.: *LINPACK Users' Guide*. SIAM, Philadelphia, PA, 1979.

[11] Dongarra, Jack J.; Du Croz, Jeremy; Hammarling, Sven; and Hanson, Richard J.: An Extended Set of FORTRAN Basic Linear Algebra Subprograms. *ACM Trans. Math. Soft.*, vol. 14, no. 1, 1988, pp. 1–17.

[12] Dongarra, Jack J.; Du Croz, Jeremy; Hammarling, Sven; and Hanson, Richard J.: ALGORITHM 656. An Extended Set of Basic Linear Algebra Sub-

programs: Model Implementation and Test Programs. *ACM Trans. Math. Soft.*, vol. 14, no. 1, 1988, pp. 18–32.

[13] Dongarra, Jack J.; Du Croz, Jeremy; Hammarling, Sven; and Duff, Iain: A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Soft.*, vol. 16, no. 1, 1990, pp. 1–17.

[14] Dongarra, Jack J.; Du Croz, Jeremy; Hammarling, Sven; and Duff, Iain: ALGORITHM 679. A Set of Level 3 Basic Linear Algebra Subprograms: Model Implementation and Test Programs. *ACM Trans. Math. Soft.*, vol. 16, no. 1, 1990, pp. 18–28.

[15] Anderson, E.; Bai, Z.; Bischof, C.; Demmel, J.; Dongarra, J.; Du Croz, J.; Greenbaum, A.; Hammarling, S.; Mckenney, A.; Ostrouchov, S.; and Sorensen, D.: *LAPACK Users' Guide*. SIAM, Philadelphia, PA, Second ed., 1995.

[16] *MATLAB Compiler User's Guide*. The MathWorks, Inc., Natick, MA, October 1996.

[17] Maghami, Peiman G.; Kenny, Sean P.; and Giesy, Daniel P.: *PLATSIM: A Simulation and Analysis Package for Large-Order Flexible Systems (Version 2.0)*. NASA TM 4790, 1997.

[18] Belvin, W. Keith; Maghami, Peiman; Tanner, Cheri; Kenny, Sean; and Cooley, Vic: Evaluation of CSI Enhancements for Jitter Reduction on the EOS AM-1 Observatory. *Dynamics and Control of Large Structures. Proceedings of the Ninth VPI&SU Symposium*, L. Meirovitch, ed., Virginia Polytechnic Institute and State University, Blacksburg, VA, May 1993, pp. 255–266.

[19] Giesy, Daniel P.; and Armstrong, Ernest S.: *FREQ: A Computational Package for Multivariable System Loop-Shaping Procedures*. NASA TM 4127, 1989.

[20] Chiang, Richard Y.; and Safonov, Michael G: *Robust Control Toolbox. For Use with MATLAB*. The MathWorks, Inc., Natick, MA, August 1992.

[21] Govaerts, W.; and Pryce, J. D.: Mixed block elimination for linear systems with wider boarders. *IMA Journal of Numerical Analysis*, vol. 13, no. 2, 1993, pp. 161–180.

[22] Householder, Alston S.: *The Theory of Matrices in Numerical Analysis*. Blaisdell Publishing Company, New York, 1964.

Table 1. Time (seconds) to Calculate Frequency Response Function Using Original Implementation

| System size[a] | Freq. vector length | Feed-forward present | New MEX-code | New M-code | Old FOR-TRAN code | Old M-code |
|---|---|---|---|---|---|---|
| 1/1 24+39 | 200 | No | 1.98 | 11.18 | 1.26 | 7.57 |
| 1/1 24+39 | 200 | Yes | 2.47 | 13.50 | 1.10 | 7.62 |
| 1/1 24+39 | 2000 | No | 24.78 | 108.87 | 10.08 | 71.18 |
| 1/1 24+39 | 2000 | Yes | 19.60 | 135.20 | 10.36 | 71.92 |
| 3/5 184+39 | 200 | No | 5.03 | 18.53 | 29.84 | 287.93 |
| 3/5 184+39 | 200 | Yes | 3.85 | 14.18 | 27.50 | 290.25 |
| 3/5 184+39 | 2000 | No | 46.08 | 147.00 | 213.69 | 2818.78 |
| 3/5 184+39 | 2000 | Yes | 35.77 | 188.93 | 200.09 | 2830.32 |
| 10/27 1406+39 | 200 | No | 60.40 | 231.25 | 5714.94 | 49846.62 |
| 10/27 1406+39 | 200 | Yes | 73.77 | 227.40 | 5756.99 | 49199.20 |
| 10/27 1406+39 | 2000 | No | 591.50 | 1922.75 | 24928.01 | - |
| 10/27 1406+39 | 2000 | Yes | 615.73 | 1897.22 | 24929.14 | - |

[a] In "m/n" in the first line, "m" is the number of inputs and "n" is the number of outputs.
In "m+n" in the second line, "m" is the number of structural states and "n" is the number of controller states.

Table 2. Time (seconds) to Calculate Frequency Response Function Using Current Implementation

| Transfer function | Number of system states | New MEX-code | New M-code | Old FORTRAN code | Old M-code | GP code |
|---|---|---|---|---|---|---|
| $T_{yr}$ | 221 | 14.26 | 21.48 | 37.64 | 738.15 | 142.52 |
| $T_{yw}$ | 221 | 21.74 | 33.04 | 50.58 | 1232.54 | 153.31 |
| $T_{y^{pr}r}$ | 221 | 17.16 | 27.89 | 38.97 | 764.57 | 145.58 |
| $T_{y^{pr}w}$ | 221 | 30.00 | 45.87 | 53.93 | 1278.97 | 158.49 |
| $T_{\epsilon r}$ | 221 | 14.47 | 21.50 | 35.88 | 738.83 | 142.71 |
| $T_{ur}$ | 221 | 13.46 | 20.23 | 36.52 | 731.16 | 142.49 |
| $T_{ud}$ | 221 | 14.48 | 22.03 | 27.37 | 368.50 | 129.09 |
| $T_{yr}$ | 1445 | 27.27 | 46.97 | 5218.06 | - | 1413.02 |
| $T_{yw}$ | 1445 | 83.86 | 142.43 | 5859.32 | - | 1550.30 |
| $T_{y^{pr}r}$ | 1445 | 45.37 | 87.68 | 5436.88 | - | 1415.65 |
| $T_{y^{pr}w}$ | 1445 | 141.63 | 257.97 | 6052.90 | - | 1551.37 |
| $T_{\epsilon r}$ | 1445 | 27.34 | 47.78 | 5377.79 | - | 1407.49 |
| $T_{ur}$ | 1445 | 22.05 | 42.72 | 5196.56 | - | 1422.89 |
| $T_{ud}$ | 1445 | 30.58 | 51.21 | 4664.98 | - | 1240.30 |

| REPORT DOCUMENTATION PAGE | *Form Approved* *OMB No. 07704-0188* |
|---|---|

| | | |
|---|---|---|
| 1. AGENCY USE ONLY *(Leave blank)* | 2. REPORT DATE<br>March 1998 | 3. REPORT TYPE AND DATES COVERED<br>Technical Publication |

| | |
|---|---|
| 4. TITLE AND SUBTITLE<br>Algorithms for Efficient Computation of Transfer Functions for Large Order Flexible Systems | 5. FUNDING NUMBERS<br>632-10-14-06 |
| 6. AUTHOR(S)<br>Peiman G. Maghami and Daniel P. Giesy | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)<br>NASA Langley Research Center<br>Hampton, VA 23681-2199 | 8. PERFORMING ORGANIZATION REPORT NUMBER<br>L-17686 |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)<br>National Aeronautics and Space Administration<br>Washington, DC 20546-0001 | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER<br>NASA/TP-1998-206949 |
| 11. SUPPLEMENTARY NOTES | |
| 12a. DISTRIBUTION/AVAILABILITY STATEMENT<br>Unclassified–Unlimited<br>Subject Category 18 Distribution: Standard<br>Availability: NASA CASI (301) 621-0390 | 12b. DISTRIBUTION CODE |

13. ABSTRACT *(Maximum 200 words)*

An efficient and robust computational scheme is given for the calculation of the frequency response function of a large order, flexible system implemented with a linear, time invariant control system. Advantage is taken of the highly structured sparsity of the system matrix of the plant based on a model of the structure using normal mode coordinates. The computational time per frequency point of the new computational scheme is a linear function of system size, a significant improvement over traditional, full-matrix techniques whose computational times per frequency point range from quadratic to cubic functions of system size. This permits the practical frequency domain analysis of systems of much larger order than by traditional, full-matrix techniques. Formulations are given for both open- and closed-loop systems. Numerical examples are presented showing the advantages of the present formulation over traditional approaches, both in speed and in accuracy. Using a model with 703 structural modes, the present method was up to two orders of magnitude faster than a traditional method. The present method generally showed good to excellent accuracy throughout the range of test frequencies, while traditional methods gave adequate accuracy for lower frequencies, but generally deteriorated in performance at higher frequencies with worst case errors being many orders of magnitude times the correct values.

| | |
|---|---|
| 14. SUBJECT TERMS<br>Large flexible structures, Controlled structures, Transfer functions, Bode plots, Efficient computation | 15. NUMBER OF PAGES<br>74 |
| | 16. PRICE CODE<br>A04 |

| | | | |
|---|---|---|---|
| 17. SECURITY CLASSIFICATION OF REPORT<br>Unclassified | 18. SECURITY CLASSIFICATION OF THIS PAGE<br>Unclassified | 19. SECURITY CLASSIFICATION OF ABSTRACT<br>Unclassified | 20. LIMITATION OF ABSTRACT |